TAMPERE UNIVERSITY OF TECHNOLOGY

SALUM ABDUL-RAHMAN

The Effects of Open Source License Properties on Software Architecture

Master of Science Thesis

Examiner: Tommi Mikkonen
Examiners and topic approved by the
Council of the Faculty of Computing
and Electrical Engineering
9th of April 2014

TIIVISTELMÄ

TAMPEREEN TEKNILLINEN YLIOPISTO
Tietotekniikan koulutusohjelma
Abdul-Rahman, Salum: Avoimen lähdekoodin lisenssien ominaisuuksien vaikutukset ohjelmistoarkkitehtuuriin
Diplomityö, 67 sivua, 5 liitesivua
Elokuu 2014
Pääaine: Ohjelmistotuotanto
Tarkastaja: Tommi Mikkonen
Avainsanat: avoin lähdekoodi, avoimen lähdekoodin lisenssit, ohjelmistotuotanto, ohjelmistoarkkitehtuuri, tekijänoikeusrikkomus

Avoimen lähdekoodin lisenssien avulla ohjelmistokehittäjät voivat yhteistyössä toisilleen tuntemattomien kehittäjien kanssa jatkokehittää ja levittää ohjelmistoja maksamatta erillistä rahallista korvausta. Avoimen lähdekoodin lisenssit voivat kuitenkin olla vaikeaselkoisia ja haitata ohjelmiston hyödyntämistä kaupallisesti sekä eri lisenssien ominaisuudet voivat olla ristiriidassa keskenään. Nykyiset lisenssien hallintamenetelmät eivät ota huomioon kaikkia avoimen lähdekoodin lisenssien ominaisuuksia ja komponenttien todellisen tekijänoikeuksien varmistaminen voi olla vaikeaa. Kaikki ohjelmistokehittäjät eivät uskalla käyttää avointa lähdekoodia, koska eivät ymmärrä avoimen lähdekoodin lisenssien ominaisuuksia tai niiden hallintamenetelmiä.

OSSLI-tukimusprojektissa kerättiin systemaattisen kirjallisuuskatsauksen avulla tietoa tieteellisen tutkimuksen nykyisestä käsityksestä avoimen lähdekoodin lisenssien vaikutuksista ohjelmistotuotantoon. Tämä diplomityö muodostaa kirjallisuuskatsauksen löydösten, ontologioiden ja yleisen systeemisteorian avulla kehyksen, jolla hahmotetaan avoimen lähdekoodin lisenssien ominaisuuksien vaikutuksista ohjelmistoarkkiehtuuriin. Tämä OSSLI-kehys rakentuu abstraktista ja sovelletusta laista, ohjelmistoarkkiehtuurista, ohjelmistokehityksestä, liiketoiminnasta ja sosiaalisesta verkostosta sekä huomioi myös lisenssien ominaisuudet.

Diplomityössä arvioidaan OSSLI-kehyksen avulla avoimien lähdekoodien lisenssien riskien hallintaan käytettyjen työkaluja ja menetelmiä sekä kuvataan miten tutkimusprojektissa kehystä käytettiin uuden ohjelmistoarkkitehtuuritason lisenssienhallintatyökalun kehittämiseen. OSSLI-kehys osoitti hyödyllisyytensä avoimen lähdekoodin lisenssien ominaisuuksien vaikutusten ymmärtämiseen.

ABSTRACT

TAMPERE UNIVERSITY OF TECHNOLOGY
Master's Degree Programme in Information Technology
Abdul-Rahman, Salum: The Effects of Open Source License Properties on Software Architecture
Master of Science Thesis, 67 pages, 5 Appendix pages
August 2014
Major: Software Engineering
Examiner: Tommi Mikkonen
Keywords: Open source, open source licenses, software engineering, software architecture, copyright violation

Open source licenses enable software developers to co-operate with unknown developers to modify and redistribute software without direct financial costs to themselves. Detecting the actual licenses and copyright holders of open source components can be difficult and open source licenses can conflict with each other and can make profiting from open source difficult. Current license compliance methods do not take into account all open source license properties. Some developers are afraid to use open source, because they do not understand open source license properties or license management methods.

In the OSSLI project current understanding of the different effects of open source license properties on software engineering was gathered by a systematic literature review. This thesis uses the results of the literature review, ontologies and general system theory to construct a framework to show how the properties of open source licenses affect software architecture. This OSSLI framework consists of the abstract legal system, procedural legal system, software architecture system, software engineering system, business system and social system.

This thesis uses the OSSLI framework to evaluate current methods and tools to manage open source license issues and shows how the OSSLI framework was used in the research project to design a new tool to manage open source license compliance through software architecture. The OSSLI framework showed its utility in understanding the effects of open source license properties.

PREFACE

> ”Think lightly of yourself and think deeply of the world.”
> -Miyamoto Mushashi

I would like to thank Adjunct Professor Imed Hammouda for giving me the opportunity and guidance to participate in academic open source license research. A big thank you is also extended to the other members of the TUT Open Source Research Group, Antti Luoto, Alexander Lokhman and Terhi Kilamo, for their comradeship and feedback in our pursuit of knowledge and wisdom. Thank you to Professor Tommi Mikkonen for helping me finish what I started. Thanks to Henri Tanskanen for his legal expertise. I will also express my gratitude towards Tampere University of Technology, Aalto University, Tekes, Validos, HH Partners, Symbio, Tekla, and Wapice for supporting the OSSLI project that enabled the research that led to this thesis.

Thank you Virve for everything.

Salum Abdul-Rahman
In Tampere on the 23rd of June 2014

CONTENTS

ABBREVIATIONS AND TERMINOLOGY

Abbreviations

ACTA	Anti-Counterfeiting Trade Agreement
AGPL	Affero Gnu Public License
ASLA	Automated Software License Analyzer
ccREL	Creative Commons Rights Expression Language
DCT	Dependency Checker Tool
FLOSS	Free and open source software
FSF	Free Software Foundation
FUD	Fear, Uncertainty & Doubt
GPL	Gnu Public License
GST	General system theory
IEC	International Electrotechnical Commission
IEEE	Institute of Electrical and Electronics Engineers
IP	Intellectual property
IPR	Intellectual property rights
ISO	International Organization for Standardization
JSON	JavaScript Object Notation
KIF	Knowledge Interchange Format
KRO	Knowledge Representation Ontology
LKIF	Legal Knowledge Interchange Format
LOC	line(s) of code
ODRL	Open Digital Rights Language
OSI	Open Source Initiative
OSL	Open source license
OSSLI	Advanced Tools and Practices for Managing Open Source Software Licenses project
RDF	Resource Description Framework
SEI	The Carnegie Mellon Software Engineering Institute

SEO	Software Engineering Ontology
SPDX	Software Package Data Exchange
SWEBOK	Software Engineering Body of Knowledge
TRIPS	Agreement on Trade Related Aspects of Intellectual Property Rights
WIPO	World Intellectual Property Organization
WTO	World Trade Organization
W3C	World Wide Web Community and Business Groups
XML	Extensible Markup Language

Terminology

attitude (legal)	Describes the mental relationship of a legal person with a norm or action or between norms
copyleft	a clause in license that prevents licensed software being combined with licenses with additional requirements
epistemology	A theory of what knowledge is
expression (legal)	Is a conveyance of a proposition in medium
mereology	The study of what constitutes a whole an its parts
norm	A rule that defines whether something is allowed, required or prohibited
open source license	A software license that allows modification and redistribution of the source code for free
open source software	Software licensed with an open source license
proposition	A claim which may be true or false
qualification	A judgement on whether a norms or claims are true or false or contradictory
reciprocal license	An open source license with a copyleft clause

1. INTRODUCTION

> Anakin: ”What has that got to do with anything?”
> Yoda: ”Everything! Fear is the path to the dark side. Fear leads to anger. Anger leads to hate. Hate leads to suffering. I sense much fear in you.” [1]

This thesis shows how we can understand open source license properties and manage risks related to open source licenses using software architecture design. This introduction explains what the perceived benefits and risks of open source licenses are and why software architecture is a possible tool for helping to manage them. The goals of this thesis are discussed along with its structure.

1.1. Motivation

There are numerous benefits to be gained from using open source software components in software development. These benefits vary from the availability of free high quality components to the open bazaar model of development [2]. Although there are no direct financial costs of using open source components, there are financial risks stemming from the possibility intellectual property rights violations and risk of loss of trade secrets by being forced to publicly release source code. Both these fears and and benefits are direct results of the terms of open source licenses.

Software architecture is used to help design, build, and evaluate software systems. There are many definitions for the term software architecture varying from the abstract split of a system into various functional components to the documentation describing these relationships. This thesis uses the definition offered by the ISO/IEC/IEEE standard 42010-2011 [3] which defines the architecture as the “fundamental concepts or properties of a system in its environment embodied in its elements, relationships, and in the principles of its design and evolution.“ Open source software licenses can be linked to software components which appear in software architecture as concepts. These open source

components can be found on multiple levels of software architecture, so it is possible to analyze the effects of open source license on the software architecture level through these concepts, properties and relationships in different environments.

Current research in open source licenses and software architecture does not cover their relationships completely. By reviewing current peer reviewed literature covering these subjects we can gather a holistic view of the interactions between open source licenses and software architecture.

The benefits of using open are shrouded by fear of being forced to divulge all source code linked to open source. Raymond [4] claims that this is due to a Microsoft FUD (Fear, Uncertainty, Doubt), a marketing strategy designed to confuse the copyleft clause of the GNU General Public License (GPL) and GNU Lesser General Public License (LGPL) with all open source licenses. All open source licenses do not have a copyleft clause. Since anybody can create a new open source license the exact terms vary by license and can lead to the copyleft terms being worded in many ways and being activated in different conditions. Because of this variance in the conditions and license, uncertainty over whether an open source license contains a copyleft clause and how copyleft clauses work, fear of open source licenses persists

This research has been conducted in the scope of the "OSSLI - Advanced Tools and Practices for Managing Open Source Software Licenses" project. The research goals of the OSSLI project were to develop a better understanding of open source licensing concerns, study the best practices for open source license compliance and identify well known solutions to open source licensing problems. Based on this knowledge the project's goal was to develop a new tool for license compliance for software design and architecture evaluation.

1.2. Objectives

The goal of this thesis is to systematically describe the interaction between open source licenses and software architecture based on current research. This description will be formulated by classifying concepts using General system theory and ontologies. The Ontologies used will be John F. Sowas' Knowledge Representation Ontology [5], the LKIF-ontology [6], and the Software Engineering Ontology by Wongthongtham et al [7].

The formally described system will be used to evaluate how tools and methodologies for managing open source license concerns are used in software architecture development. By evaluating how current practices take in to account different interactions between open source licenses and software architecture, we can show how well risks and benefits related to using open source licenses can be considered during software design. The evaluation can help to identify relationships between open source licenses and software architecture that create value or risk. We attempt to identify tools possible risks and benefits that are not taken into account in current tools and methodology. This thesis shows how the framework was used to develop tools in the OSSLI project in order to understand how they can help during the software engineering process.

These findings can be used to evaluate the benefits and risks of using open source components and help prevent unintended license breaches and encourage the use of open source components when beneficial. By collecting the current understanding of open source licenses the overcome the fear, uncertainty and doubt related to the complexity of open source license and software development.

1.3. Structure

Chapter 2 presents the background of open source licenses and software architecture and their roles in software engineering. In addition Chapter 2 presents the research methodology applied in this thesis. Chapter 3 presents the framework into which the findings of the literature review concerning the current understanding of open source license and software architecture is mapped. Chapter 4 attempts to define the connections between open source licensed and the abstract legal domain to software architecture and software engineering based on the findings of the literature review. Chapter 5 presents The OSSLI framework that is developed based on the findings of chapters 3 and 4. Current tools and methods used to manage open source licenses reduce the risk of intellectual property rights violations are presented and the OSSLI framework is used to evaluate their role in the software engineering process. Chapter 6 shows how the framework was used to design the OSSLI tool developed during the OSSLI research project for open source license analysis. Chapter 7 presents an evaluation of the framework and benefits of using it to evaluate tools and methods of license management. Also the methodology applied

is evaluated and potential weaknesses and improvements are identified. Chapter 8 draws conclusions based on the findings presented in the earlier chapters, evaluates the benefits of these findings to research, and identifies further possible areas of research.

2. RESEARCH BACKGROUND AND METHODOLOGY

> "The Way of the carpenter is to become proficient is the use of his tools, first to lay his plans with a true measure and then to perform his work according to plan." [8]

This chapter presents the background of open source licenses and software architecture and how the research questions of this thesis were defined. Then the chapter describes how the research methods, Systematic Literature Review, General System Theory and Ontologies, are used to answer gather and organize information to answer the research questions.

2.1. Background

According to ISO, IEC and IEEE [9] the term software engineering has two definitions:

> "1. the systematic application of scientific and technological knowledge, methods, and experience to the design, implementation, testing, and documentation of software."
> "2. the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software."

There are many benefits to be gained by using open source in software development, but there are also risks involved. Software engineering means taking systematic and quantifiable approach to these issues. This section presents some of the expected risks and benefits of using open source software components in software development and defines open source licenses and software architecture. These expected risks and benefits form

the basis for the motivation and the research methodology used in this thesis and the hypothesis that software architecture can be used manage license concerns.

The obvious benefit of using open source components is that there are no direct financial costs [10, 11, 12]. Open source components are often developed with reusability in mind making them easy to integrate [12, 13, 14]. Some research has also shown that open source components can exceed and are often of comparable quality to proprietary components when it comes to security and the number and nature of defects [10, 11, 13]. The accessible nature of open source components can make finding developers familiar with the components in question easier [13].

Bahn [15] identifies three risks associated with using open source components: 1) upstream intellectual property concerns, 2) viral software issues, and 3) non-infringement warranties or intellectual property (IP) indemnity issues. Upstream intellectual property issues refer to cases where an open source component contains code that has been included without respecting the rights of the copyright holders. Viral software issue refers to the fear that a company would be forced to release its software as open source if it contains a component with an open source. Lack of warranties and indemnity issues refer to the fact that using contrary to traditional business models open source users are not protected by the the basic warranties related to standard business practice. If there is a problem or lawsuit the open source user will have to defend themselves and can not rely on the software provider for help. McGowan [16] also states that the enforceability of open source licenses is arguable especially depending on jurisdiction. Both McGowan [16] and Bahn [15] state that the risks are hard to evaluate due to lack of case law related to open source licenses.

2.1.1. Open Source Licenses

Rosen [17] defines a license as the legal way a copyright and patent owner grants permission to others to use his intellectual property and an open source license as the way a copyright and patent owner grants permission to others to use his intellectual property in such a way that software freedom is protected for all. An another way of defining an open source license is using the Open Source Definition(OSD) defined by the Open Source Initiative (OSI) [18]. OSI is an non-profit organization that maintains the Open Source

Definition and evaluates whether a license fulfils the requirements of the OSD [17]. The open source definition has 10 clauses, but Perens [18] identifies three rights that OSD is designed to ensure [18]:

- The right to make copies of the program, and distribute those copies.
- The right to have access to the software's source code, a necessary preliminary before you can change it.
- The right to make improvements to the program

These three rights are the software freedoms referred to by Rosen [17].

Open source license have been traditionally organized into reciprocal licenses and academic licenses [17]. Reciprocal licenses contain a copyleft clause that is designed to ensure that the covered software is only used in open source software systems. Due to the basic rights granted by open source licenses, they in practice also contain a patent grant to the implementation in the software. Some licenses like the Apache Public License 2.0 (APL) make this grant explicit [19]. There are also patent licenses and content licenses that imitate open source software licenses in their attempt to make their covered content free and modifiable. [17]

2.1.2. Software Architecture

The Software Engineering Body of Knowledge [20] (SWEBOK) places software architecture as a part of software design as described in Figure 2.1 SWEBOK divides the software design process into architectural design and detailed design. SWEBOK identifies that there are multiple definitions of software architecture some relating to the way a software system is organized and descriptions of this organization. [20]

ISO/IEC/IEEE standard 42010-2011 defines the architecture of a system as the fundamental concepts or properties of a system in its environment embodied in its elements, relationships, and in the principles of its design and evolution [3]. This makes the architecture and abstract entity on its own that is related to its system. Fowler [21] defines software architecture as making design decisions. This fits nicely in with SWEBOK associating software architecture with design, but is more closely related to ISO/IEC/IEEE standard 42010-2011 definition of architecting. The Software Engineering Institute at

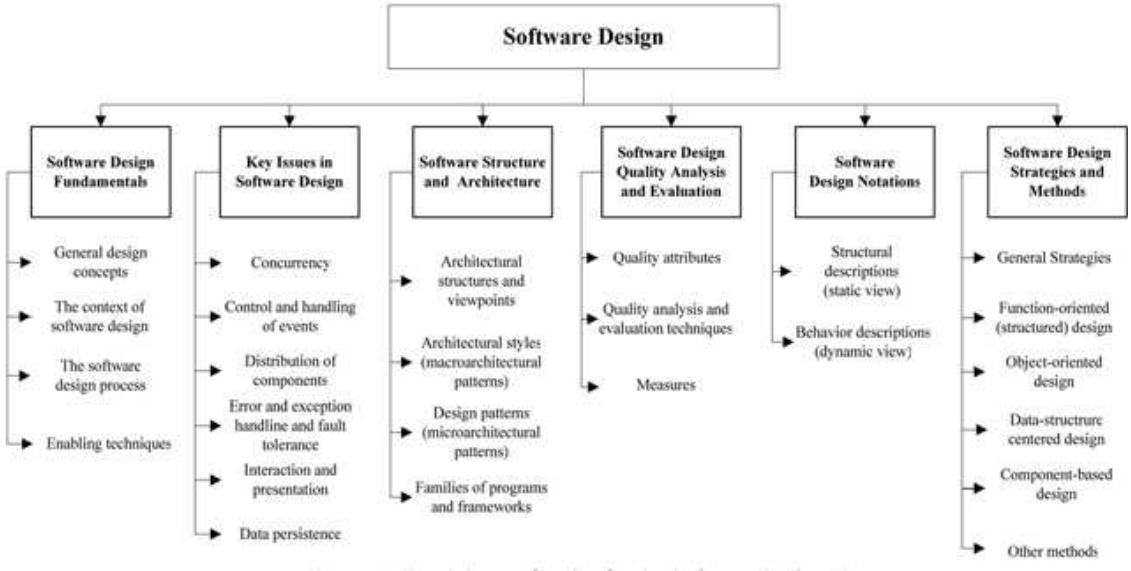

Figure 2.1: Breakdown of topics in Software Design knowledge area. [20]

Carnegie Mellon (SEI) describes software architecture as: "The software architecture of a program or computing system is a depiction of the system that aids in the understanding of how the system will behave [22]." This description corresponds with ISO/IEC/IEEE standard 42010-2011 definition of architecture description.

SWEBOK allows all three possible definitions and all views equate to a knowledge of the software system. Hislop [23] presents two epistemologies: the objectivist and practice based view. The objectivist believes knowledge is an entity of itself whereas the practice based view holds that knowledge is only apparent in the way it affects the actions of a person or a an organization. Fowler's view on architecture makes his view of architecture a member of the practice-based epistemology. The SEI and ISO/IEC/IEEE definitions fall firmly in the realm of the objectivist epistemology though distinction between them is clear. The SEI definition finds knowledge only exist when it is codified in the architecture description while ISO/IEC/IEEE definition allows the the knowledge of the system defined as the architecture to exist as abstract entity. Since Fowler's and SEI's definitions match the terms defined by ISO/IEC/IEEE standard 42010-2011 this thesis uses the ISO/IEC/IEEE terminology.

Licensing issues with software do not limit themselves to the software architecture level. However licensing issues related to open source licenses do not usually extend to the software system level where interconnections between software components are

handled by the operating system or network services. When analyzing open source license violations at a level more detailed than software architecture, we enter into territory of code duplicates and copied code. Detecting copied code is beyond the scope of this thesis due to it being complex enough to warrant its own research, but we do take into account the methods that can be used on an architectural level to evaluate license violations once licensed code is found.

2.2. Research Questions and Methods

The research questions this thesis answers are threefold. First: What are the relationships through which open source licenses affect software architecture? Based on this the second research question is: How do the tools and methods used to manage open source licenses affect the relationships between software architecture and open source licenses? The third question is: Can this knowledge be used design tools to help manage license issues in architecture.

In order to show that software licenses affect software development at an architectural level it is necessary to show how exactly this interaction manifests itself. In order to develop an understanding of the extent of interaction between software architecture and software licensing we need to gather data on the issue and then present this data in a scientific model. This research data has been collected in the form a literature review of peer-reviewed sources. An attempt is made to place the findings of the literature review into a hierarchical system complex based on the General System Theory and by identifying the components and relationships of this system in an ontology. This system can be analyzed for deficiencies or extended by including information from non peer-reviewed sources in order to develop further research questions into the nature of interactions between the fields software engineering and law as well as software business and intellectual capital.

2.2.1. Systematic Literature Review

We performed a literature review based on the methodology presented by Brereton et al [24]. We developed a review protocol that was used to find peer reviewed sources on open source licenses and legal or intellectual property right related issues.

Based on the protocol we developed a Boolean structured query to represent our interests:

```
("Open Source" OR OSS OR FOSS OR FLOSS OR "Free Software")
AND
(License OR Licenses OR "Legal Issues" OR Copyleft
 OR "Immaterial Rights" OR "Intellectual Property"
 OR "Software Patents")
```

We then inputted this query into several academic databases and tried to find publications that dealt with the issue of open source licenses. The databases searched are listed as follows:

```
ACM Digital Library, http://dl.acm.org/
IEEExplore, http://ieeexplore.ieee.org/
JSTOR, http://www.jstor.org/
Wiley Online Library, http://onlinelibrary.wiley.com/
```

We selected those articles that seemed relevant based on their abstracts and that had been cited at least three times or were published during or after 2010. The criteria of relevance was that the abstract suggests a link between the legal aspects of open source software licenses and the software engineering domain. The citation count was based on the number of citations given to the considered articles by Google Scholar. We included the newer publications without need for citation, because we wanted to include a view of current research and considered requiring citations from newer articles could exclude relevant sources. In addition we selected all articles from one peer reviewed journal, the "International Free and Open Source Software Law Review", because it was the only journal that specializes in issues of open source law. By definition all articles are relevant for the study and because it is the only journal it is also the leading journal. These findings were classified using the presented ontologies and organized in relation to each other to form the system models.

2.2.2. General System Theory

According to Skyttner [25] one of the core concepts of General System Theory (GST) is hierarchy. Hierarchy in general system theory refers to the concept that systems are made up of subsystems. Skyttner [25] also presents George Klirs mathematical model for

system in which a system is group of things and their relations. According to von Bertalanffy [26] a system consists of components and relationships between components that form a definable whole. These components and relationships can themselves be analyzed as systems thus becoming the subsystems that make up a lower level of the hierarchy. Klirs model is so abstract that it can to applied to almost any phenomena [25]. The com-

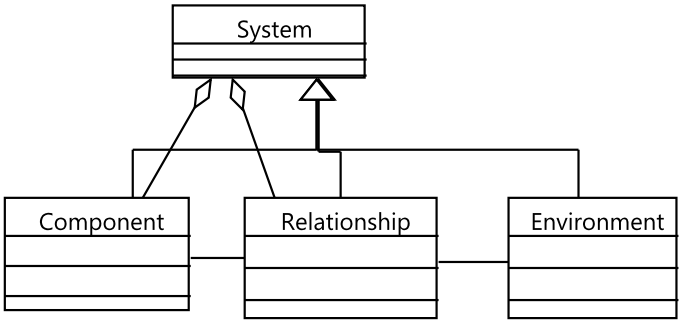

Figure 2.2: The General System Theory in UML-notation

bination of being able to include multiple levels of phenomena from different domains as long there is some interconnection means that General System Theory can be used to describe multidisciplinary phenomena and according to von Bertalanffy [26] the need for this kind of flexibility in scientific research was a motivator in the development of General System Theory. Due to the fact that software engineering as a science has no direct connection to legal sciences, a multidisciplinary approach is needed in order to understand the connection between software architecture and software licenses. Defining what makes a component or relationship a part of a system or subsystem is dependent on whether the system in question can be considered a whole with or without the component [26].

Skyttner defines a conceptual system as an organization of ideas expressed in symbolic form [25]. Software architecture is a conceptual representation of a software system usually expressed in documentation. Software architecture can thus be considered as a conceptual system. A software license is a document defining what kind of interaction is necessary between the licensor and licensee in order for the licensee to perform some actions on or with the software. Thus a software license can also be considered a conceptual system.

According to Skyttner conceptual systems consist of abstract concepts, but they can affect physical actions [25]. In this sense both legal documents and software architecture

documentations represent conceptual systems. They both describe abstractions and affect how people operate.

The connection between a software system and a software license is straightforward, but complexity ensues when a software system is made up of multiple components with different licenses. Licenses operate within the scope of a legal system, whereas software architecture is developed in a software development system. Software development happens in an organization like a university or software company and by individuals who in turn are governed by a legal system. The purpose of the software development is to produce a program or software system that in turn serves multiple functions; usually to provide some benefit to the developers in the form of automation or help with some sort of task or perhaps enjoyment or fiscal or social benefit. All of these factors have to be able to be taken into account in the system model.

These interconnections seem straightforward, but they are split among multiple levels of system hierarchy and represent multiple levels of ontological concepts. It is therefore necessary define what are the actual concepts in these system are and what is the nature of their relationships and what is the hierarchy of the systems. In addition to systems modelling it is therefore necessary classify each component in the different systems according to an ontology.

2.2.3. Ontologies

Gruber [27] states that "A conceptualization is an abstract, simplified view of the world that we wish to represent for some purpose." and that "An ontology is an explicit specification of a conceptualization." In order to be able to express the different components in a system we need to have some way to define these components. By using an ontology to describe the components we can be sure that concepts are not confused. Natural languages can be imprecise as words can have multiple meanings and different terms can be used to refer to the same concept. In order to avoid semantic confusion it is necessary to define all the concepts and their relations ontologically.

Ontologies can be divided into top level ontologies and domain specific ontologies. Top level ontologies are broad attempts to define structure of ontological concepts from a broad point of view or to provide tools to merge other ontologies [28]. As we attempt

to analyze subjects that are different fields of study it is useful to use domain specific ontologies to identify the concepts used. A top level ontology is also required to map show similarities and dissimilarities between concepts in the different ontologies as well the findings of literature review. In this case we chose Sowa's Knowledge Representation Ontology as the top level ontology and LKIF as the legal ontology and the Software Engineering ontology by Wongthongtham et al [7] for software engineering.

Noy and Hafner [28] split top level ontologies according to their top level taxonomies as either hierarchical division into more exact components or distinction approach which several dimensions are simultaneously used to define the properties of a concept. Sowa's Knowledge Representation ontology belongs to the latter type, with the top level concepts drawn from classification along the axes of Physical and Abstract into Continuant and Occurrent classes and on a third axis of Independent, Relative and Mediating classes [5]. Both LKIF and the Software engineering ontology belong to the hierarchical approach in which more general components are divided into more detailed instances on each descending level [6, 7]. By using a different system of taxonomy at the top level than in the domain specific ontologies we attempt to avoid possible incongruencies and conflicts between disjoint rigid hierarchies as well as providing a more holistic understanding of the domain, by providing additional perspective to the classification system.

Table 2.1: Matrix of Sowa's Ontologies three axis and central categories. [5]

	Physical		Abstract	
	Continuant	Occurent	Continuant	Occurent
Independent	Object	Process	Schema	Script
Relative	Juncture	Participation	Description	History
Mediating	Structure	Situation	Reason	Purpose

LKIF is an extension of of the Knowledge Interchange Format ontology (KIF) and therefore shares the same top level ontology as KIF. KIF is a top level ontologyand was considered as anoption for our top level ontology. This would have been simpler, but using John F. Sowa's Knowledge Representation Ontology or KRO, an ontology based on distinction approach as defined by Noy & Hafner [28], as top level ontology would provide more perspective for research. It would additionally provide a chance of mediation if the concepts from SEO would not map to the LKIF and KIF.

Knowledge Representation Ontology

The top level of the KRO consists of three axes. One axis is is divided into physical and abstract. Physical objects or processes must have mass or energy. They have to either exist as physical objects or impact physical objects directly. Abstract concepts are ideas or processes that are not perceivable outside human thought process; examples include platonic forms and human constructs like friendship and freedom. On the temporal axis concepts are divided between occurrent and continuant. Continuant concepts are the same over a subjective time interval whereas occurrent concepts can defined by their time interval. On the relational axis concepts are either independent, relative or or mediating. Independent concepts are self defined, they exist in and of themselves. Relative concepts relate two or more concepts to each other. Mediating concepts bring other concepts into relationships with each other. Whereas a person is an independent concept, a marriage is a relationship between two people, yet a family is mediating concept that can include multiple people with multiple mediating relationships to each other.[5]

The second level of the KRO consists of concepts that are intersections of concepts from the relational axis and the abstract-physical axis. This creates group of six concepts each concept inherits the all of the properties of both their parent concepts. The final top level of KRO consist of concepts that are subclasses of concepts from the second level and temporal axis [5]. A second level concept could therefore be a Nexus which has the properties of both Physical and Mediating, something concrete which is defined by being a groups of concrete concepts like a forest is defined as grouping of trees. On third level a Nexus is divided into a Structure and a Situation depending on the temporal axis. A Occurrent Nexus is is Situation like a cab ride consisting of passengers and driver and vehicle but existing only for a finite amount of time. Structure being Continuant could be a crossroad that connects multiple roads, and this defining property does not change over the applied time period. All seven top level concepts, six second level concepts, and 12 third level concepts and their hierarchy are shown in Figure 2.3.

LKIF

The top level of the LKIF-core ontology consist of Mental Concept, Occurence, Physical Concept and Abstract Concept. LKIF-core is split into multiple modules that cover differ-

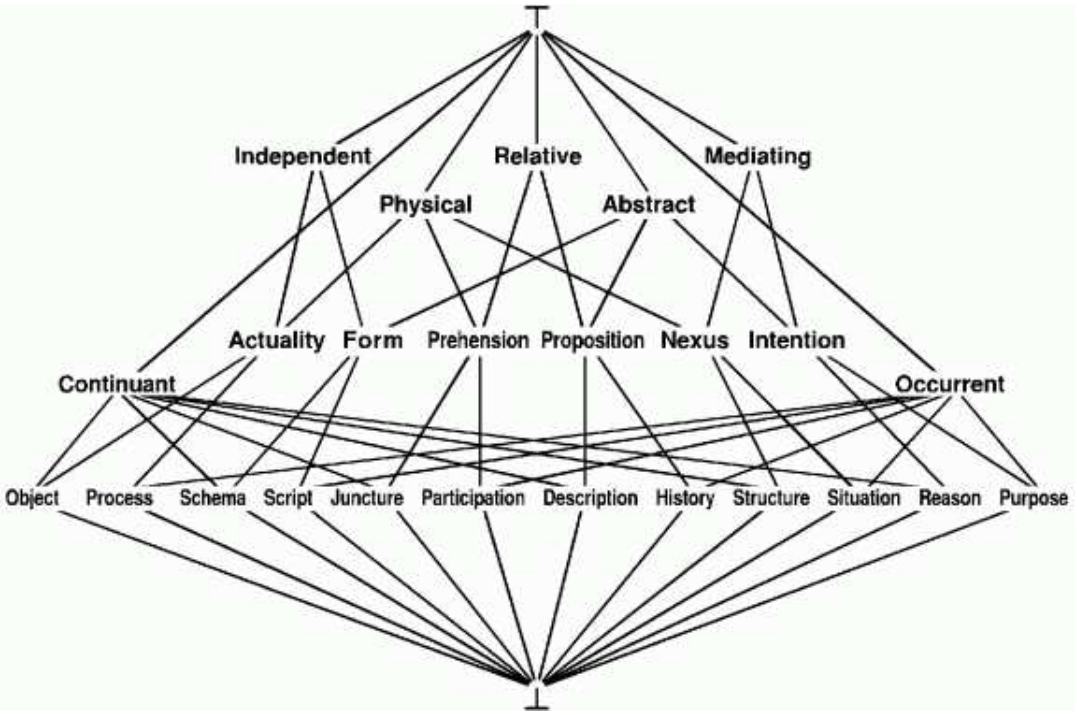

Figure 2.3: Sowa's hierarchy of top-level categories. [29]

ent aspects. In addition to the top module there are other modules that cover very general concepts: Place, Mereology, Time, Change, Actions, Agents, Organizations, Roles. Of particular interest are the modules that cover specific legal concepts. These modules are Propositions, Attitudes, Expressions, Qualifications and Norms. The top levels of the LKIF-core taxonomy are displayed in Figure 2.4 [6].

The top level concepts of LKIF map mainly to the top level elements of Sowa's ontology. LKIF is designed as an extension of KIF which is a top level ontology. Because both KIF and the KRO are top level ontologies many higher level concepts appear in both. The detailed relationships are described in Table 2.2

By studying Figure 2.5 we can see that all of the top level categories have a top class from the the highest levels of Sowa's hierarchy except for physical objects. At this level LKIF is very abstract. Due to its nature of concentrating on legal issues, which are often abstractions, this abstraction persists to most of the ontology.

Software Engineering Ontology

The hierarchy of top classes of SEO are described in Figure 2.6. For some reason, the core of the ontology has been grouped under the Software Engineering Domain class. This

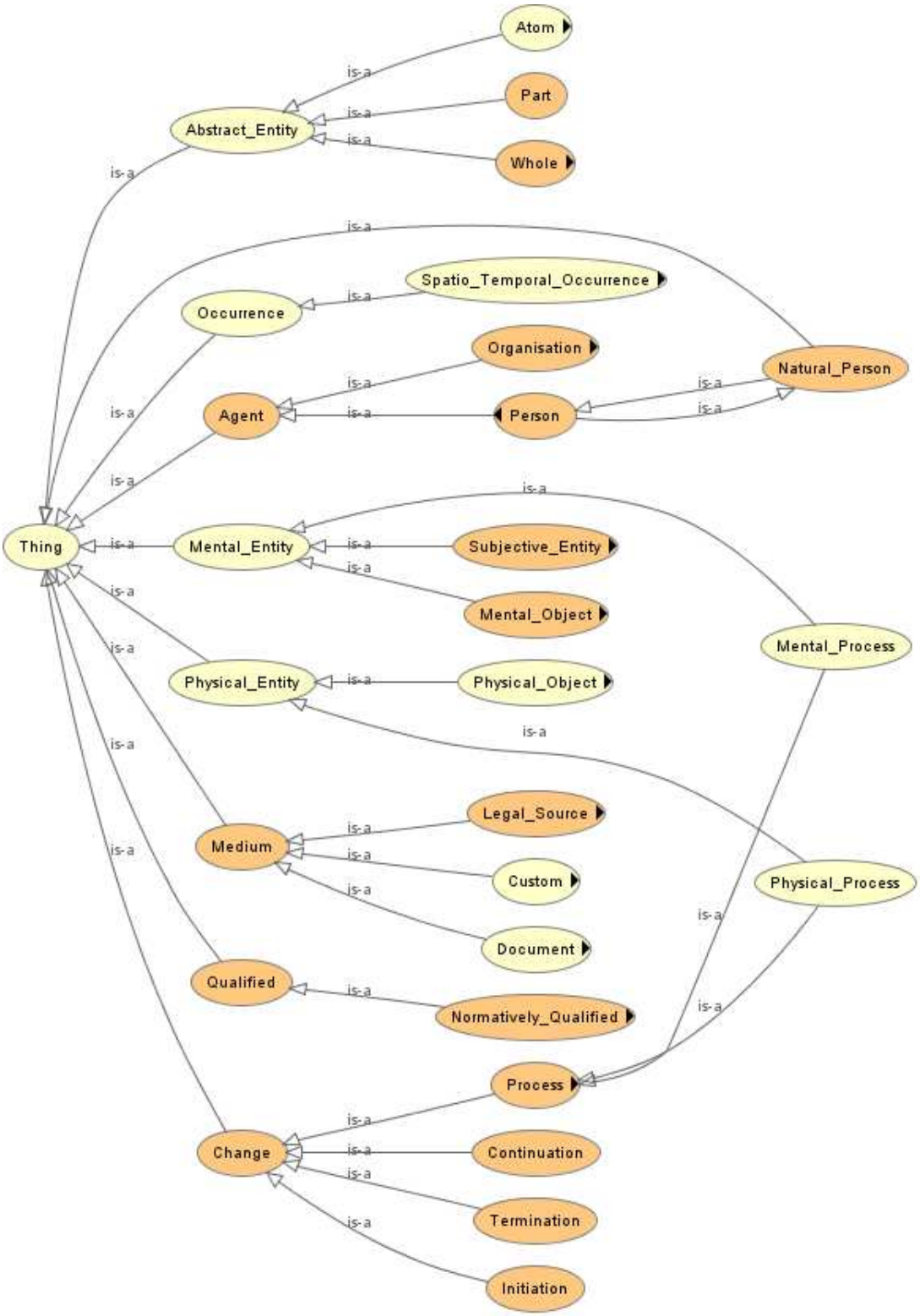

Figure 2.4: Taxonomy of LKIF-core ontology.

Table 2.2: Top level concepts in LKIF ontology and their superclasses in Sowa's ontology

Parent class in Sowas ontology	Subclass in LKIF-core ontology	Description
Abstract	Abstract_Entity	All abstract entities are Abstract.
	Mental_Entity	A mental entity has no physical from so it is an abstract entity.
Continuant	Interval	An interval defines a time interval an continuant concept is defined by its time interval.
Independent	Atom	An Atom is indivisible and can defined by itself
Intention	Qualified	Being qualified by a qualification is an abstract mediating relationship
Juncture	Relative_Place	A juncture is a connection in space and Relative_Place is defined by it's connection to another physical place
Mediating	Change	Achange is defined by the different states that are part of the change.
	Plan	A plan is defined byt the subplans or actions it consists of.
	Composition	A Composition is defined by its multiple parts.
	Oganization	An Organiaztion is defined by its multiple individuals.
Object	Natural_Person	A natural person is a physically existing human being.
	Absolute_Place	Absolute_Place is defined by its physical location in space.
	Person	A person is physically existing human being, equivalent class of Natural_Person.
Occurrent	Natural_Person	Natural_Persons only exist for a period of time.
	Occurence	An occurence is has an temporal beginnning and an end.
Physical	Place	Place has a location wich is a physical property.
	Spatio_Temporal_Occurrence	A Spatio_Temporal_Occurrence has physical properties.
	Physical_Entity	Physical entities are physical.
Relative	Subjective_Entity	Subjective entity is defined in relation to to an other entity.
	Medium	Mediums bear expressions and are defined by their relationship to the message as well as senders and receivers.
	Qualified	A qualified entity is defined by its relation to a qualification.
	Agent	Agent relates and entity to a role.

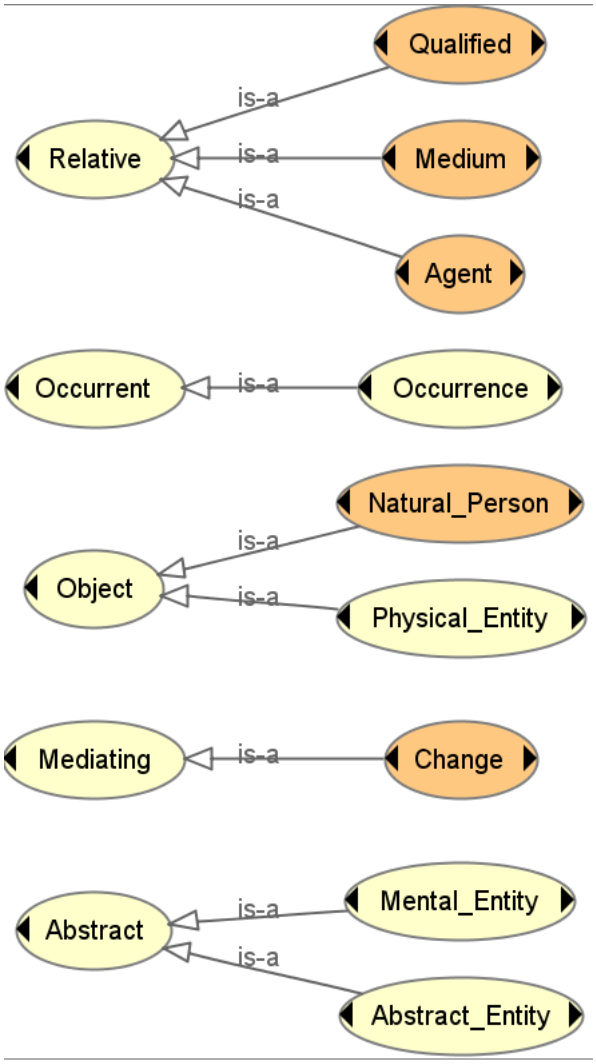

Figure 2.5: LKIF top level classes mapped as subclasses to Sowa's hierarchy of top-level categories.

choice is understandable due to the narrow nature of the SEO, but it would be extremely difficult to use Sowa's top classes consistently or accurately if one were to assigned as a top class for the Software Engineering Domain. The Software Engineering Domain class is ignored and we assign top classes for its subclasses from Sowa's ontology. The subclasses of the software Engineering domain are Software Design, Software Construction, Software Tools, Software Testing and Software Requirements. How the subclasses of The Software Engineering domain and the other top level classes of SEO, Reputation, People and software engineering project map to KRO is described in Table 2.3

From Figure 2.7 we can see that the top level classes of the SEO map to lower classes of the KRO than the the classes of LKIF in Figure 2.5. The narrower scope of SEO allows using lower level top classes which helps to keep the system more exact.

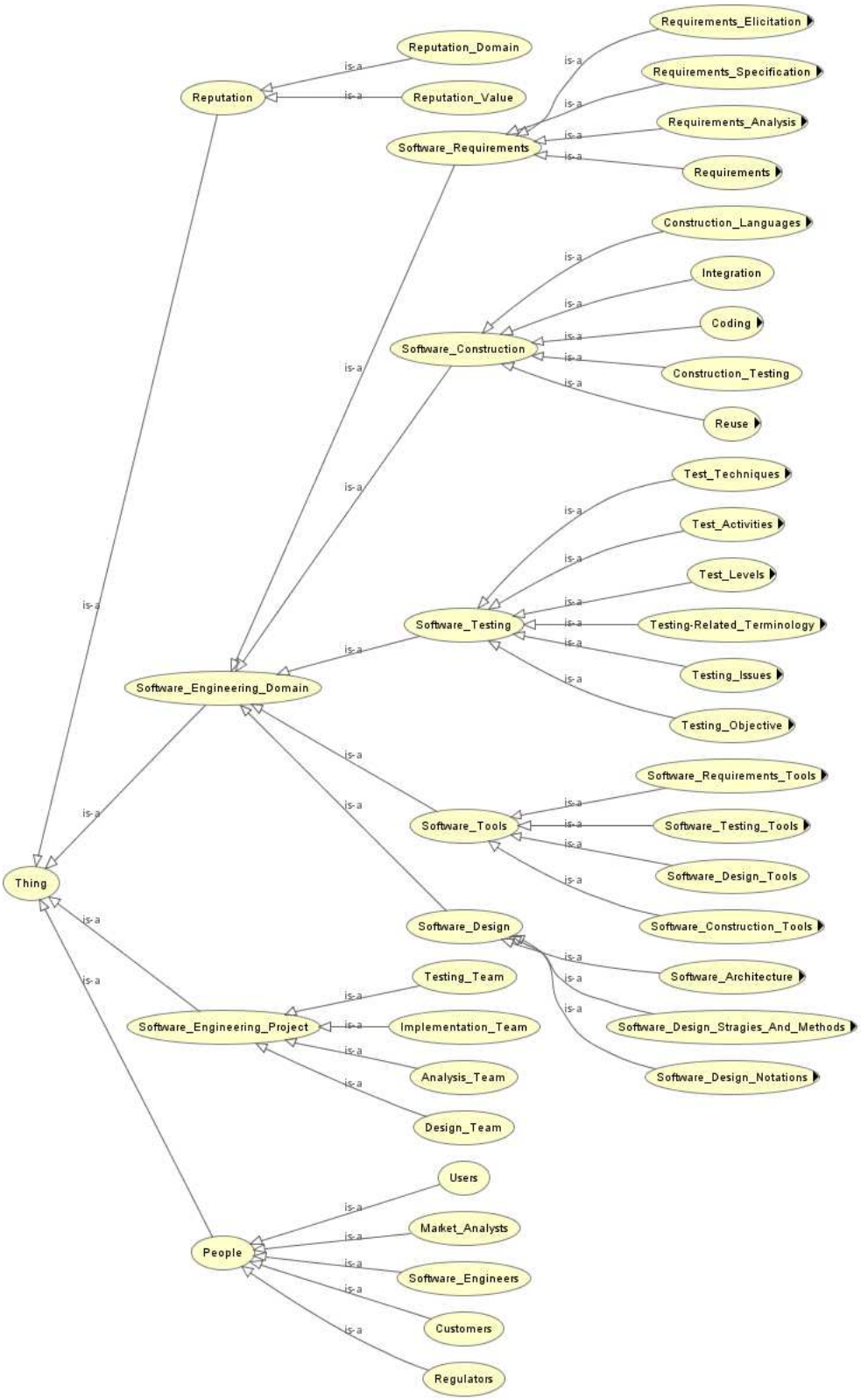

Figure 2.6: Taxonomy of SE-ontology top level.

Table 2.3: Top level concepts in software engineering ontology and their superclasses in Sowa's ontology

Parent class in Sowas ontology	Subclass in SE ontology	Description
History	Reputation	A reputation is description of previous behaviour.
Object	People	People are have a physical form and exhibit firstness and have a temporal beginning and end
Description	Software_Design	Software design is the process of making design decisions of software, these decisions manifest as documentation or direct implementation.
Physical	Software_Construction	Producing and compiling th source code an installation and integration are physical acts.
	Software_Tools	Software tools are physical tools even if they exist as programs inside a computer.
Prehension	Software_Testing	Software testing relates to the physical code to the software requirements.
Proposition	Software_Requirements	Software requirements define the relationship between the software and its function, this relationship is abstract as it can exist before the software is created.
Situation	Software_Engineering_Project	Software engineering project mediates the members of the team and their tools and the software being worked on.

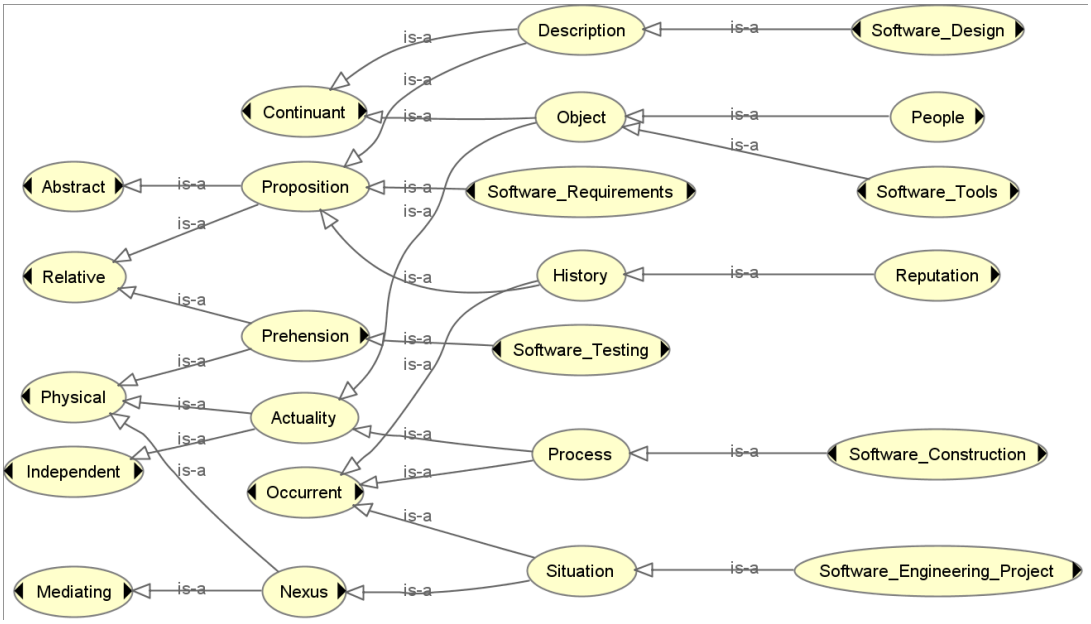

Figure 2.7: SEO top level classes mapped as subclasses to Sowa's hierarchy of top-level categories.

The weakness of the SEO becomes apparent when trying map lower level concepts to the KRO. In the SEO Software Design is defined as a process and has subclasses. Software Architecture, Software Design Notations, and Software Design Strategies and Methods. In this context Software Architecture is described as static documentation and the Notations cannot be considered as a process. The relationship here is clearly more of "is a part of" not "is a subclass of." This is a common problem faced when designing an ontology, therefore it not use full to define the KRO superclasses we impose on the top level classes as hereditary, but rather to apply them to lower level classes in the proper context when we encounter them in the research data.

Merged Ontology

From Figure 2.8 we can see that most top level classes of SEO find a superclass in LKIF. That is to be expected because the legal domain is more general than the software engineering domain. The exception of course is the natural person to people equivalence, legal matters and software engineering both need people to be meaningful.

Figure 2.9 combines the top level of the LKIF and the SEO with their superclasses in the KRO into one view.

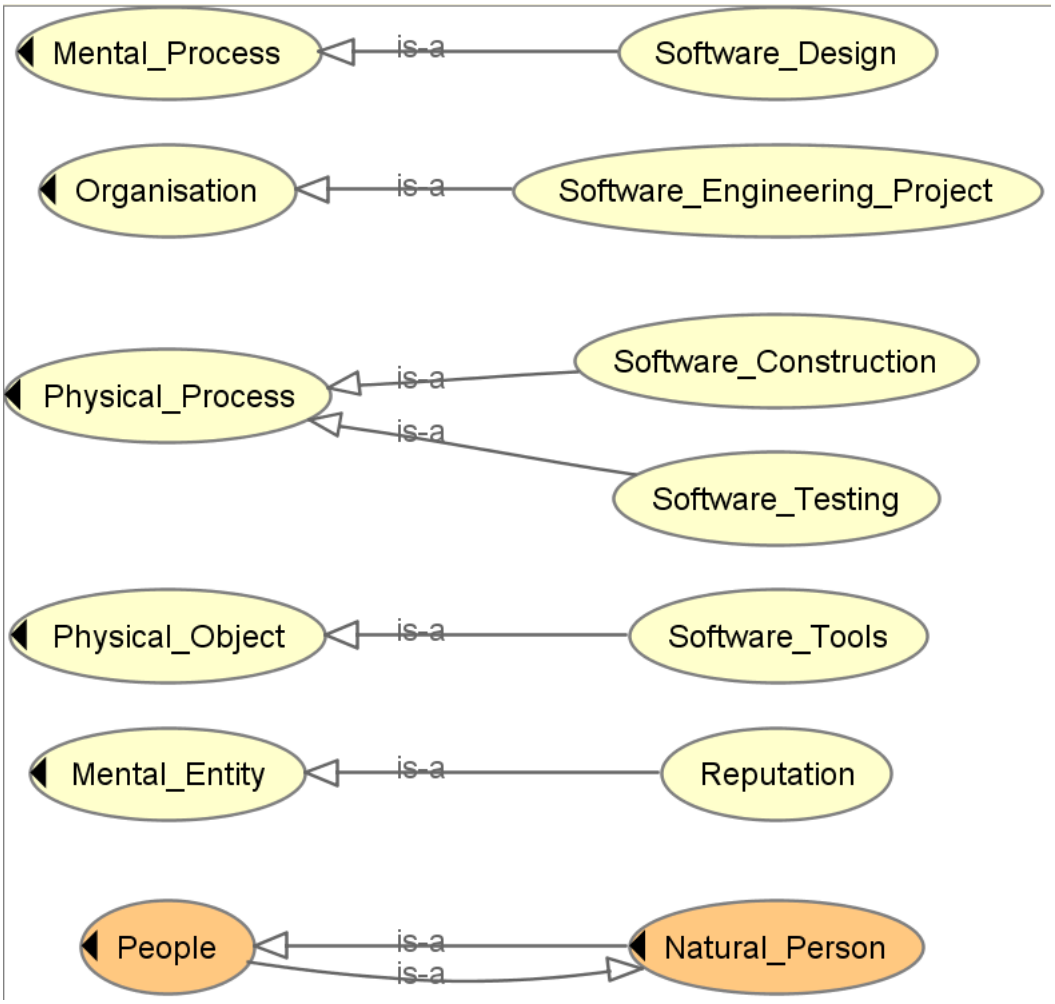

Figure 2.8: Relations between concepts in LKIF and Software Engineering Ontology.

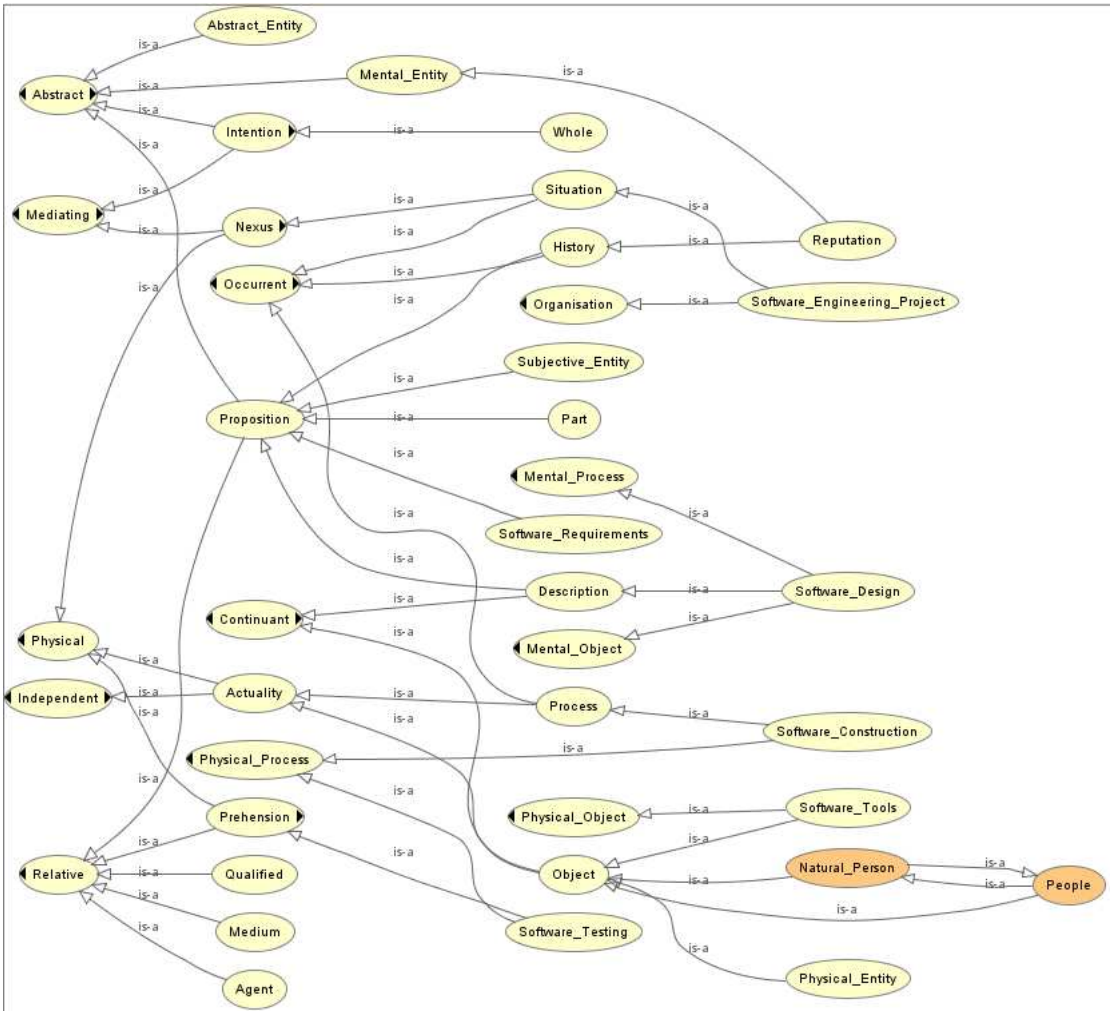

Figure 2.9: Merged top level of ontologies of KRO, SEO and LKIF. Unused classes of KRO are hidden.

3. ABSTRACT LEGAL AND SOFTWARE ARCHITECTURE SYSTEMS

> "What is called the spirit of the void is where there is nothing. It is not included in man's knowledge. Of course the void is nothingness. By knowing things that exist, you can know that which does not exist. That is the void." [8]

The premise of the research is formed by the software architecture system and the abstract legal system. Software architecture forms its own abstract system. Open sources licenses are part of the abstract legal systems. By defining these systems in terms of the ontologies we form a basis from which can explore the links between the software architecture system and open source licenses using the findings of the systematic literature.

3.1. Abstract Legal System

This section contains a description of the concepts related to the abstract legal system and their relationship to the taxonomies generated using ontologies. In legal terms a qualification expresses a judgement about something. All qualifications are relative to some expression. An evaluative qualification qualifies something as good or bad, desirable, or undesirable. Normative qualifications or norms qualify something as allowed or not allowed. Norms can be divided into absolute norms, which make something mandatory, or prohibited and rights which someone can choose to exercise like copyright. A Proposition is a legal term for an expression which can be evaluated in a legal context. Claiming that a person has a copyright over some work is a proposition. Proposition can be divided into legal expressions and evaluative propositions. Evaluative propositions are evaluatively qualified, which means that they express some qualification about the proposition. Almost any action or state that can be considered in legal terms is a a legal expression and as such a subclass of proposition. When considering the abstract legal world we have to

take into account attitudes. Legal attitudes affect how people evaluate propositions. If you infringe on exclusive rights granted to the copyright holder unintentionally and can show that you have been intentionally mislead about the copyright situation as opposed to being negligent in regard to the copyright, your attitude toward the action of infringing on copyright is different and can affect the legal evaluation of the situation.

German & Hassan [30] say that there are 6 legal structures that can affect software: 1) copyright, 2) related rights, 3) patents 4) trade secrets, 5) design rights and 6) trademarks. Trademarks can cover, in particular logos and names of companies and products. Trade secrets cover information that a company takes reasonable steps to protect. Patents, trademarks and design rights are publicly applied and declared. All software is protected by copyright, but particularly in the United States patents are used to protect software more often than in the rest of the world [31].

3.1.1. Copyright and Related Rights

Copyright is occurrent and relative to an artefact. Copyright is an obligative right which means that the copyright holder has the right to prevent other people to utilize his work. Copyright is automatically created when a novel artefact is created. It is unclear where the line goes when defining a novel work when it comes to source code. Copying a few lines of the web maybe accepted practice but German & Hassan [30] state that copying less than 100 LOC could create a derivative work meaning that the original author would have a copyright claim. In the scope of this thesis any component that is considered on the software architecture level will be assumed to be original and complete enough to warrant copyright. In cases where a component contains sections of code copied from other sources in an scope that warrants copyright, all requirements of all licenses that cover copyrighted code must be met in order to not commit copyright infringement.

When it comes to open source licenses the concepts of collective work and derivative work are important. According to Välimäki [32] the use of these terms stems from United States copyright legislation and they are used in several Open source licenses. Open source software is mainly built by many contributors and projects use components and libraries from various sources. Also the popular GPL family of licenses differentiates its terms depending on whether work is collective or not [33]. A derivative work is something

which is directly based on other work [33]. A collective work is more loosely coupled but is still distributed with foreign components that are required for it be meaningful [33]. A derivative work is directly based on other component, but if the component could easily switched to another one it could be argued that the complete work is collective and not derivative. According to Forbes [34] a database or dataset containing source code can be considered a derivative or collective work. In this case the dataset collector may have related rights to the dataset, such as sui generis database rights under EU law, but not a copyright.

A work in the public domain has no copyright holder and anybody can use it freely. A copyright holder can declare his work to be in the public domain or a work can enter the public domain because 70 years of the death of the creator have elapsed and therefore copy right ends. Originally a lot of software in the 1960's and 1970's were considered public domain, due to the hacker culture [2]. Richard Stallman developed the original GPL, because he felt that public domain did not ensure the freedom of his software as mentioned in subsection 2.1.1. by Rosen. In some cases it is very hard to trace the copyright holder of source code available on the internet, but this does not mean that the source code belongs to the public domain [31]. Public domain is more used in common law context. In Finland it is impossible for the content creator to fully place his work into the public domain until his copyright expires.

Copyright only applies in its legal domain. In Europe local legislations create the rules for copyright, but across Europe copyright is legislated according to EU law which states how local legislation affects actions in other countries. Across legal domains the mechanism defining how copyright is legislated in the form trade agreements: The Bern Convention of 1974, The Universal Copyright Convention of 1971, The Convention Establishing the World Intellectual Property Organization (WIPO) of 1967 , the Agreement on Trade Related Aspects of Intellectual Property Rights (TRIPS) of 1994 and the WIPO Copyright treaty of 1996 [31]. These treaties apply only to members of the signatories of the treaties, so there is no legal recourse for Europeans to uphold their copyright in North Korea. TRIPS sets the baseline of legal mechanisms, but other trade agreements such as the now unlikely Anti-Counterfeiting Trade Agreement (ACTA) could refine the international copyright protection mechanisms.

3.1.2. Patents and Trade Secrets

A software patent describes a certain programmatically implemented novel mechanism. This mechanism cannot be used commercially without license from the patent holder. Patents are public and it is the responsibility of users not to infringe on patent. Trade secrets can contain suitably protected ways of implementing or composing a program or a part of a program. Both mechanism are used to profit from a program and preventing others from profiting from same mechanism, but they take contradictory approaches. A patent claims I did this program this way and no one else may do so whereas trade secret is hiding said program so that others cannot copy the implementation. [32]

The core issue between open source and patents is that while most open source licenses imply or outright state they grant a patent license to all implementations contained in the licensed code, there is no way to make sure that the source code does not contain code that infringes on a patent held by a third party. When using a traditional proprietary licensed software the distributor has a vested interest in ensuring that the software doesn't infringe on third party patents. In most jurisdictions open source software doesn't infringe on a patent until it is used commercially and in practice the chance of discovery of a software patent infringement in privately used software is practically non-existent.

Trade secrets can affect software architecture, but are more likely to affect implementations such as algorithms. They are hardly relevant in the scope of open source since code that contains a trade secret is stolen, not licensed. In the case where an open source developer independently comes up with similar implementation as the trade secret, then this is not an infringement. If an employee of a company has not signed a confidentiality contract and publishes a program containing a trade secret with an open source licenses then the trade secret is void, because the company has not taken reasonable steps to protect the trade secret.

While software patents and trade secrets could affect open source, they offer no mechanism by which to evaluate or manage these on a software architecture level. Software covered by trade secrets have no public architecture to evaluate. If software patents were to require a architecture description of a program it could be used to evaluate whether another program with a sufficiently similar architecture infringes the patent or not. However if you consider architecture a documentation of design decisions instead of a design

contract then such an evaluation is not relevant. Software design patterns which are not considered patentable can be used to produce an architecture and as such a software architecture could hardly be patentable, or used to evaluate the infringement of a patent. Only in case where a program is in the same domain and perform the same function could the similarity of software architecture be used to evaluate patent infringement.

3.1.3. Design Rights and Trademarks

The function of the trademark is to help consumers to differentiate between companies and products. This allows companies to gain value by developing a brand and prevents competitors from benefiting or hurting each others. While the software and its components can be trademarked these trademarks are unaffected by the compositions or content of the software and thus do not are not affected by nor do they affect software architecture. Design rights consist of visually defined forms and actions. [32] While design rights can cover some parts of a software systems user interface, a design right covers just the design, and not the implementation of the design so the design right can not extend beyond the user interface of the system. This means that design rights do not affect the rest of the software architecture. Design rights and trademarks do cover software but are irrelevant when considering software architecture, except in cases where the design of the program are inseparable from the implementations. This small subset of programs could be considered old or very small programs like demos and compositions or old or simple computer games. In these cases you either infringe or don't but without a system architecture there are no options to use on the architecture level to evaluate or avoid the infringement.

3.1.4. Open Source Licenses

Open Source Licenses are seen as having properties of both contracts and bare licenses. This has led to a lot of legal confusion on the enforceability of open source licenses as bare licenses and contracts are covered by different legislation [17]. Traditionally open source licenses have been classified as academic licenses and copyleft licenses, but according to our literature review newer research approaches analyze licenses on a more detailed level.

Rosen [17] describes the properties of bare licenses through two examples. One is the classical case in property law, from which term originates, where a bare license is the

license granted by a shopkeeper to customers to enter his store in order to do business without fear of trespassing. The other example is the drivers license which a government issues, but it does not cause the government to be in any way beholden to for the recipients actions while driving [17]. This is interesting since the bare license expressed in the case of property law is not a part of formal law but can be considered to be an expression of Customary Law as it is defined in LKIF. Customary law is not formal, but it is considered legally relevant as actions that are customary or generally expected to be performed or avoided can not be considered illegal unless exceptionally forbidden or required by law. The example concerning the driver license is more in line with open ource license as it includes a formal declaration of a right that is not normally extended to all citizens similar to. In both cases the nature of the license is such that it places no requirements on the licensor. In the informal case of the shopkeeper, he can revoke his license by declaring the shop close or that the custom of the person in the shop is not desired, with some legal caveats such as discrimination. In the formal case laws or statutes have been defined for cases which lead to the revoking of a drivers license. In the case of the proprietary bare license, the recipients right to expect the license are limited by custom and in the case of the drivers license by any statutes defined.

Contracts on the other hand have clearly defined parties, the licensee and the licensor are both clearly stated as parties to the contract and every detail is hammered out. There are cultural differences in the interpretation of contracts. Northern European and North American contracts are usually interpreted to the letter, whereas Mediterranean and Chinese contracts contracts can be interpreted in the scope of relationship of the participants, whereas Japanese might be inclined to uphold the spirit of the contract over its details.

Originally open source licenses were analyzed taxonomically [17], but modern researchers have deconstructed open source licenses into components to be analyzed on a component level as well as through the composition of the components [35, 30, 36]. This analysis of license can be interpreted in the terms of GST [25]

German & Hassan [30] develop a metamodel for open source license as displayed in Figure 3.1. The license is a subclass of a Legal Document of LKIF [6] and the Mediating class from the KRO [5] because it combines the Grants in the model. Grants and Requirements are LKIF [6] Norms. Each grant is Qualified [6] by a Condition which

Mediates [5] the requirements for the Grant. The Grants are Norms which declare rights for the licensee and the requirements are norms that the licensee must uphold in other to keep the grants.

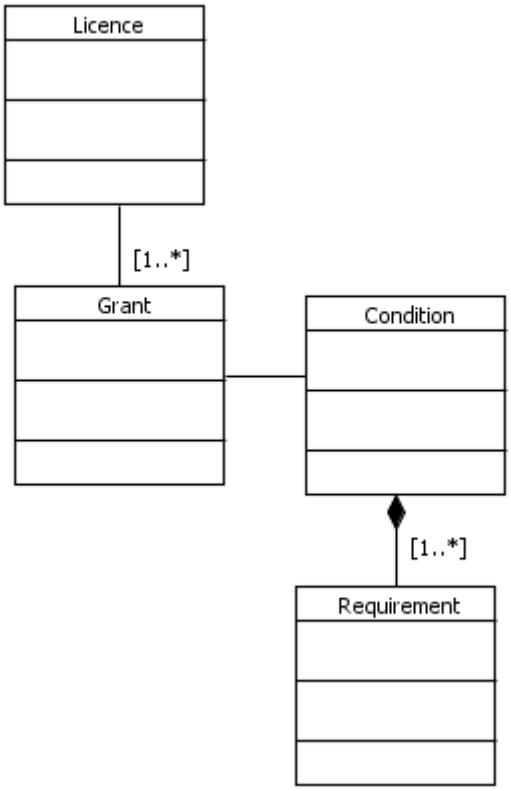

Figure 3.1: Metamodel of open source license according to German & Hassan [30].

Alspaugh et al [35] extend and modify the metamodel by German & Hassan [30] by renaming the Grant to Right and Requirement to Obligation and linking them directly to each other. As can be seen in Figure 3.2 Alspaugh et al [35] change the Condition to a Tuple which can link a number of Qualifications [6] to the Norms [6] of rights and obligations. The types of qualifications listed are the Actors, Modalities, Copyright Actions, Objects like the code covered or other works or specific other licenses [35]. An example of qualification by license reference is the MySQL FOSS License Exception, which allows JDBC drivers licensed with GPL to be used with static linking in FOSS projects as long as the license used in the project appears on a MYSQL maintained license list. Modalities can be modifiers of the domain or date or whether the right. Copyright actions are covered earlier in this subsection. Objects can be the covered code or can reference a class of code such as derivative or collective works covered created with the licensed

code.

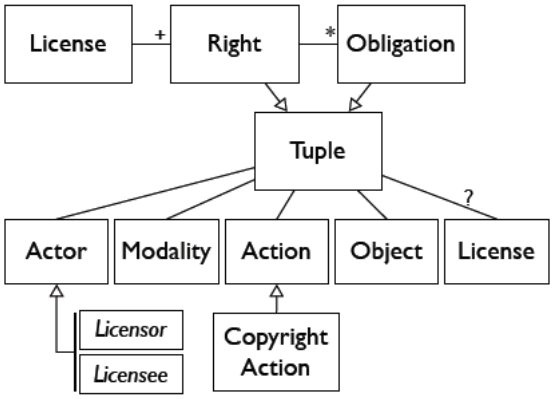

Figure 3.2: Image of meta model of open source license according to Alspaugh et al [35].

License norms can conflict, with the most obvious conflict being combining a copyleft license with a proprietary component [30, 17]. The Copyleft norm conflicts with the proprietary requirement of payment in order too use and modify code. Lesser well known is the conflict between open source licenses terms. The GPLv2 requirement of "no additional requirements" conflicts with the 4-clause BSD advertising requirement and the Apache License 2.0 patent and indemnity requirements. The composition of the license mismatch problem is presented in Figure 3.3, but it is good to note that the components and their interconnection types are concepts from the software engineering and software architecture systems.

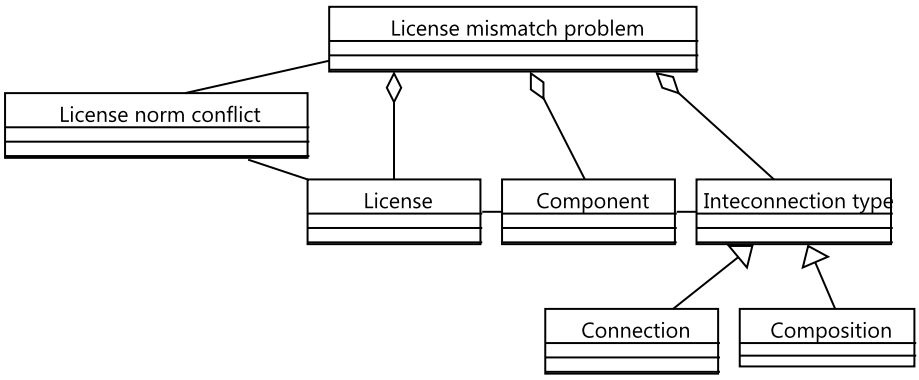

Figure 3.3: The components of the license mismatch problem.

3.2. Software Architecture System

In Section 2.1.2. we defined the three views of software architecture, and stated that we use the ISO/IEC/IEEE standard 42010-2011 [3] definition, which describes the software architecture as abstract representation of the software component organization. Regardless of interpretation software architecture is used in the design of software systems. Software architecture as a design tool affects the construction part of the software development process. According to SWEBOK [20] architecture affects high level design decisions. An architectural decision would not be which algorithm is implemented a in section of code, but how the software could be implemented so that any on member of a family of algorithms could be implemented into that section of code. The abstract components and compositions expressed in a software architecture are only relevant if they are actually implemented in the corresponding way in the actual software system.

Software architecture documentation is a symbolic representation of that abstract system which is software architecture. Software architecture documentation is a by product of the software architecting process. It can be argued that the the architecture documentation is the end product of the software architecting process, this view corresponds to the SEI view of architecture [22]. The software architecture documentation can also be viewed as a tool for communicating between the designers when designing the software architecture and as as a tool for communicating the design for those not involved in the software architecting process. The software architecture documentation along with the social interactions of the architects make it possible to communicate the abstract view of the software architecture to developers, testers, clients and others involved in the software development process. By spreading this view of the abstract architecture system the parties involved are affected by the software architecture.

4. CONNECTING SOFTWARE ARCHITECTURE AND OPEN SOURCE LICENSES

> "The thing the ecologically illiterate don't realize about an ecosystem is that it's a system. A system! A system maintains a certain fluid stability that can be destroyed by a misstep in just one niche. A system has order, a flowing from point to point. If something dams the flow, order collapses. The untrained might miss that collapse until it was too late. That's why the highest function of ecology is the understanding of consequences." [37]

Our classification and system modelling efforts have lead us to come up with four levels of systems that connect the software architecture system and the abstract legal system based on the findings of the literature review. These systems have components that appear in multiple systems and perform different roles, but these broad conceptual levels help in organizing our approach. The four levels are: 1) procedural legal level, 2) business process level, 3) software engineering level, and the 4) social level. The procedural legal level consists of the legal system which applies these abstract legal concepts to the actual persons and companies as well as the political frameworks that lead to the creation of laws and contracts. The business process level consists of the organizations that put together software engineering teams and the clients and complementary organizations that buy, use or redistribute the software. The software engineering level consists of software developers and their tools and resources. The social level is linked within all three previously mentioned levels and consist of all people involved and their relationships and communication.

4.1. Procedural Legal System

In modern societies the procedural legal system consists of two separate systems. The legislative system creates laws which are basically complex systems of norms. The judicial

system is then charged to evaluate whether suspect actions by members of society conflict with these norms and in cases where such conflict is found to assign punishment according to laws. If a criminal violation is suspected this suspicions along with any evidence are presented to the representative of the procedural legal system, like a police officer or prosecutor. The representative then gathers and evaluates the information available with an option to gather more information based on her own evaluation of the situation. Based on the evaluation of this information the conflict is either discarded or presented to the next level of system until it reaches a level where the final judgement can be made. At many points people involved in the dispute can file complaints which are then evaluated and can affect the process.

According to Välimäki [32] license conflicts are processed in civil legal process in which the proof of non compliance or compliance is up to the parties involved. Such cases begin by notification of violation to the suspected offending party by the copyright holder and are usually decided by guided negotiations instead of official judgement. Only copyright infringement that is premeditated or due to gross negligence or which performs a great harm on the copyright holder will be processed in a criminal process [32].

Public accusations of infringement may even be illegal [32]. Just the act of public suspecting a copyright violation can affect the business, software engineering and social systems, due to fear of repercussions or just hurt feelings. Legal procedures can affect the software engineering system mainly by requiring time from developers who could be working on the program instead of presenting it in court or to investigators, but proceedings can also negatively affect developer motivation and concentration. The direct effects of the legal process affect the business level in the form of damages assigned and the amount of labour and finances required in the proceedings. The outcome can affect the software engineering process if the judgement includes a choice to either uphold the copyleft requirement of a license or desist use of the licensed software, which may force software developers to change components in the system, if for some reason they are unwilling to comply with the license. These relationships are depicted in Figure 4.1.

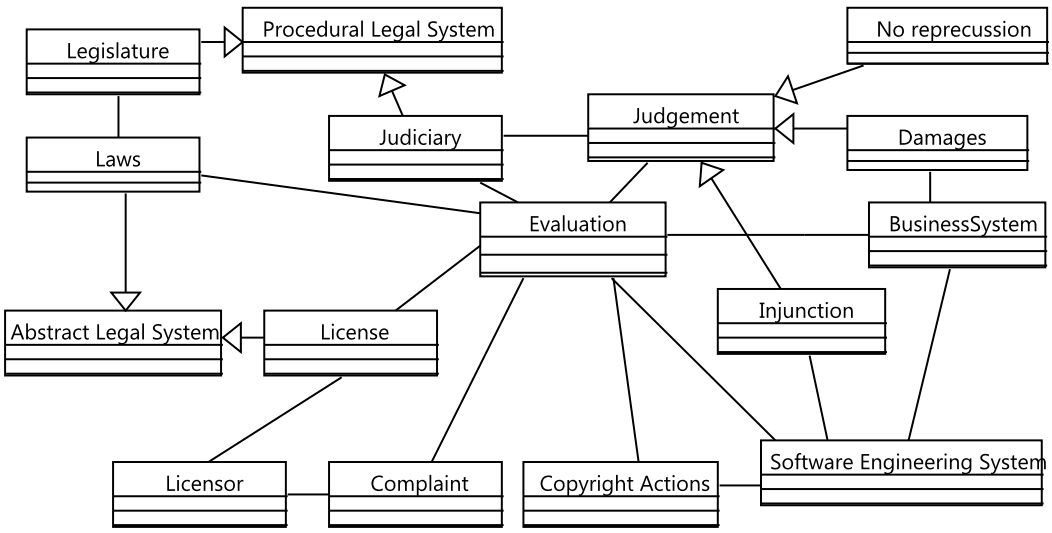

Figure 4.1: The procedural legal system in case of copyright violation and how it relates to the software engineering and business systems.

4.1.1. National Legal Systems

National legislative systems create and update national copyright legislation. They should follow the norms set by their corresponding federal legislation and ratified international copyright treaties. National legislatures are subject to lobbying as well as expert opinion when formulating copyright legislation. National judicial systems interpret national law and custom when evaluating copyright conflicts. Evaluation by national judicial systems will be affected by federal legislation or international treaties if they conflict with national legislation. Decisions made on national level can be appealed to federal jurisdiction.

4.1.2. Federal Legal Systems

Federal legislative systems like the European Parliament and Commission and the United States Congress and Senate make and update federal copyright law. Federal law is usually held as the highest source of norms. European legislature can be considered quasi federal as directives are used to harmonize national legislation, but not necessarily as legal norms [38].

Some federations like Germany have a federal judiciary, but it functions as a national judiciary. In the EU federal courts only handle cases that have been processed by national legislature and are appealed to to the federal court. Decisions made in the federal courts

can directly affect the legislation and judicial system on a national level. National legislatures have to follow federal interpretations of law, because in practice not following federal interpretations would lead to an automatic appeal to the federal level system. [38]

In North America and Britain, evaluation is based on case law which means that evaluating copyright infringement is based on law and to decisions made by previous judges on similar cases. In continental Europe evaluation is based on common law which means that the prime sources for evaluation is the law as written and the current situation as well as the judicial's own evaluation.

4.1.3. International Processes

International treaties form a framework according to which national legislature should be harmonized [38, 39]. In practice this means that if copyright infringement is performed in a different jurisdiction than in which the copyright holder operates, the copyright holder has the right to make a claim for his copyright in the judicial system which governs the place where the copyright infringement took place [39]. There are of course problems that arise from going through legal proceedings in foreign country. Basic differences such as language, culture and lack of social standing or network can form insurmountable or very expensive obstacles. In case of violations GPL family licenses the local Free Software Foundation affiliate (if existent) will probably try to help licensors from foreign countries trying to uphold their rights [40].

4.2. Business Process System

Software is usually developed in organizations. Formal organizations such as open source foundations, universities and companies develop a lot of software. This requires resources such as developers, access to computers and networks, physical space and supporting roles for the organizations. A lot of open source software is developed mainly by individuals. Himanen [41] identifies the role of large organizations as providing the required stability for large projects. Even Linux was developed originally in the shelter of Helsinki University, but it is now supported by The Linux Foundation [41]. While software is also developed by individuals with their personal tools, particularly open source components, it is important to identify that software often requires various resources to develop. These

resources are traditionally classified as financial or tangible resource or capital and immaterial resources referred to as intellectual capital. Intellectual capital is particularly important in software development as the value of software correlates with how complicated the software is. A complicated software program can be used to reduce the how complicated the related human work is. Thus the usefulness and business value of the program corresponds with how complicated the program is. Producing complicated software is a complex task and requires not only skilled individuals but allowing the skilled developers to interact in productive way. This framework for interaction is a part of the organizational capital [23]. The resources needed for software development in an organization and their relationships with each other are shown in Figure 4.2.

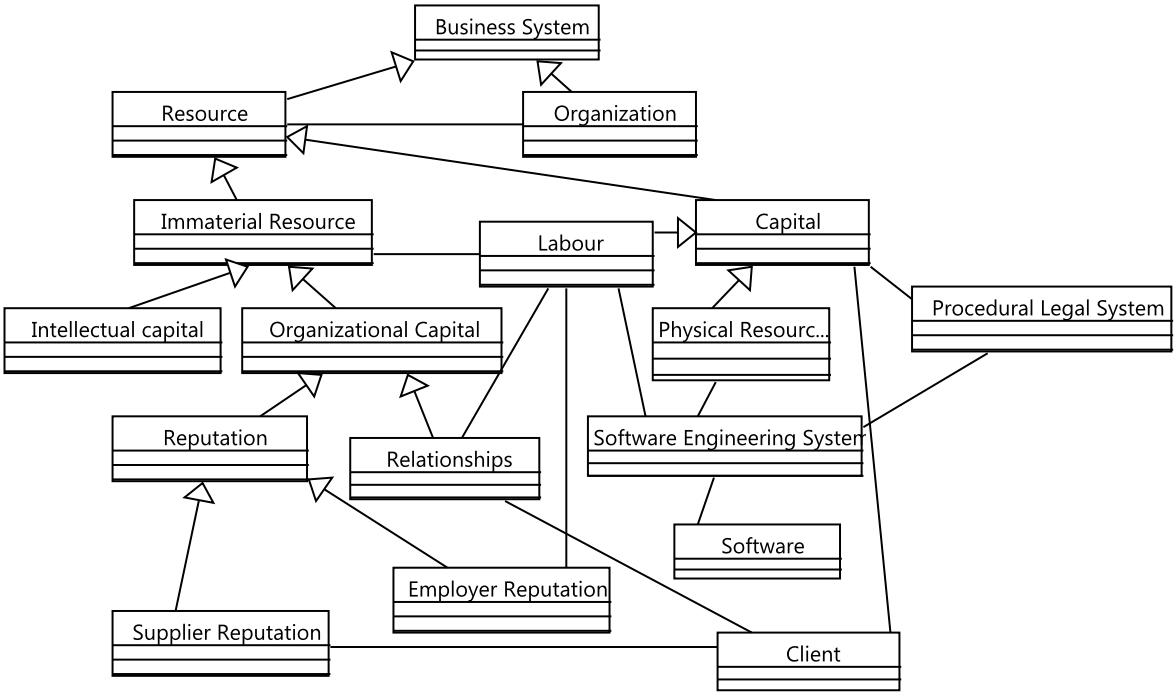

Figure 4.2: The business system of an organization developing software for a client.

Businesses using open source software benefit from community contributions to open source software projects. The form of community contributions, can be tangible such as code or immaterial capital such as a good reputation among prospective employees and clients due to association with popular software [42]. In particular using and contributing to projects with reciprocal licenses can be seen as altruistic and attract customers and co-developers [43]. On the other hand some customers fear reciprocal licenses which may

lead to a negative reputation among those clients [43]. Some communities view academic licenses as more altruistic, so when starting a new open source project the type of license and community must be considered carefully.

When dealing with open source components, not all business models are appropriate. By taking into account the business model used for the software many problems of open source license can be mitigated [43, 13]. Some companies merely incorporate open source components into their software and either sell them with a proprietary license or offer them as a service [44, 45]. The open source version may lack features of the proprietary version. Some companies offer the core of their product as open source, but offer proprietary plugins or services to support the product [44, 45]. Open source mobile applications are sold by Google and Apple services, probably because the majority of Android and iPhone users lack the technical skill to compile and install the software themselves. Many software based services use open source components because they have used them to gather other valuable assets that combined with their software they can provide a differentiated service. Hardware developers also use open source since their business value is generated from the hardware and the software complements it [44]. Competitors can use their open source components in embedded systems, but the customizations required are of such complexity that the direct benefit to competitors is not large enough compared to direct benefits for the company.

4.3. Software Engineering System

As presented in Section 2.2.3. the software engineering domain divided into five sub domains: software requirements, software testing, software construction, software tools and the people developing the software. Their interactions are displayed in Figure 4.3

Software tools are used in software construction but can also be used in software testing. Software tools do not usually affect the license issues of the produced software or used components. Our research encountered academic speculation on effects of tools on component licensing, but the literature review did not find evidence of this.

Software Requirements are based on business requirements of the software. Software requirements define where the software is going operate and what it is going to do. Soft-

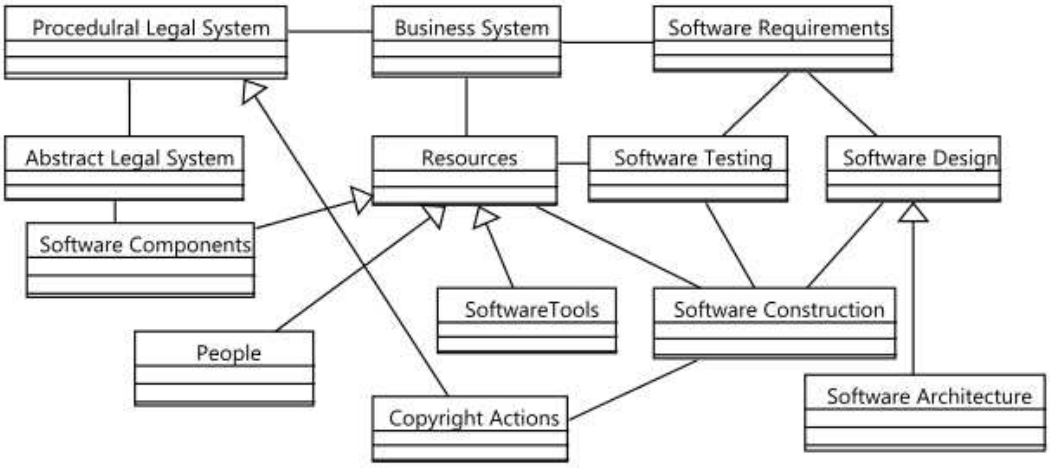

Figure 4.3: Relationships between Software Engineering systems components and Procedural and Abstract legal systems and Business System.

ware testing provides information about the system in relationship to the requirements. It is up to the developers how this information is used, usually the information is used in Software Construction but feedback can also affect design. Static and runtime analysis methods can provide information on program composition that can relate to licenses requirements.

Software design tries to answer the question how the software is constructed so that it fulfils the software requirements. Software architecture is a subset of software design. The developers use the software design to guide them in constructing the software.

Software developers are the people who actually develop the software, as such they are the people who perform all the actions in the software engineering domain. Usually they are divided into groups by function of different areas of the developed software being produced, but this is not necessary. In open source projects a developer can participate in any way she wants from just submitting a single line of code, bug report or feature request, or develop the whole program themselves. The business system affects who the developers are and what resources are available. Through the business system the procedural legal system can affect these resources. The abstract legal system affect the software components directly in the form of the software licenses. By doing software construction the developers perform actions that are considered copyright actions which can be evaluated by the procedural legal system.

How the license mismatch problem presented in Subsection 3.1. affects the software

engineering process is shown by the ontology tool for license compliance by as described Gordon [36]. Gordon uses the LKIF and describes not only the metamodel of the open source license, but also the application domain. In the QUALPSO project it was shown that license models can be used with the Carneadas system to detect license conflicts in between licenses modelled. With the application domain including system architecture, with the types on links between software components, the LKIF based OWL model can be analyzed whether the license conflicts found would affect the modelled software system [36]. The extended model can be found in Figure 4.4. This model shows how open source license properties appear in the software engineering system.

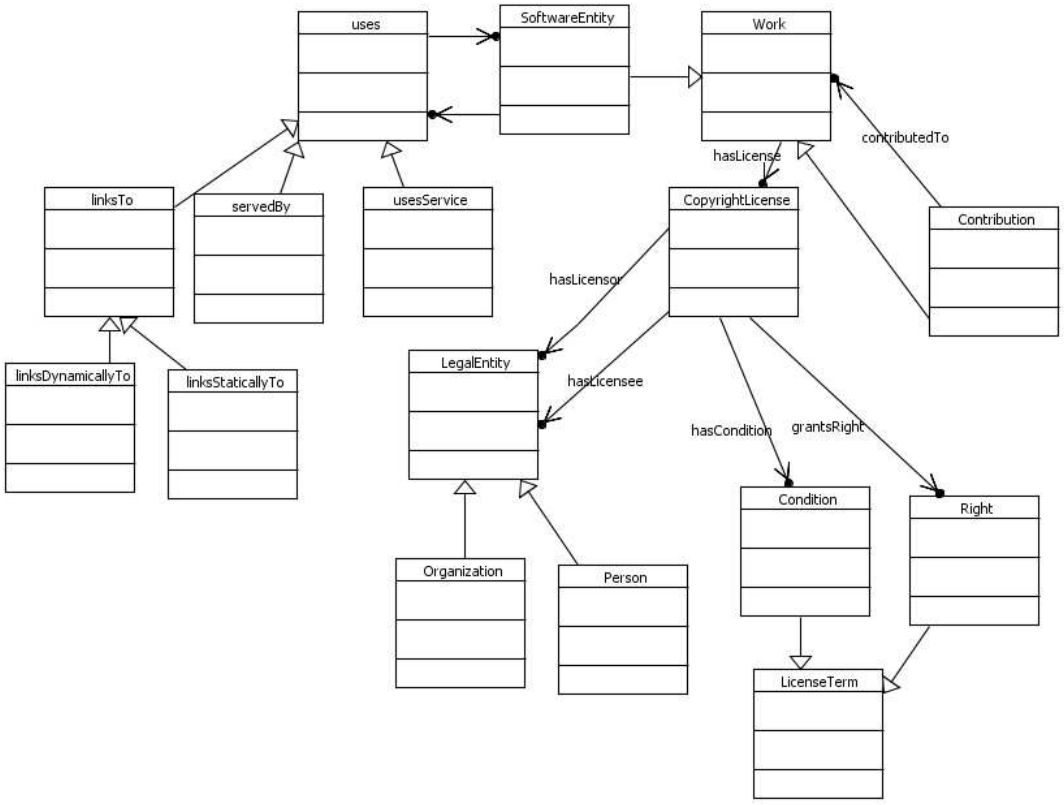

Figure 4.4: Metamodel of open source license and domain according to the Qualipso project [36]

4.4. Social System

The social system consists of the social interactions between people involved in making software happen. This includes everyone from the developers to the customers to the cafeteria staff. Software development is knowledge intensive work and as such motivation and human interaction strongly affect performance [23]. While the social system is

based on personal interactions it is beyond the scope of this work to consider individual relationships, so instead we consider the social interactions related to groups and organizations.

The relationship capital of the business system is also part social system of software producing organizations. By interacting with the open source community, organizations can affect how the open source software is developed and therefore receive greater benefits from using open source software. It is important to understand what kind of open source license is used as it affects how to interact with the community [43]. Most JavaScript is developed with permissive licenses and it could be hard to attract developers to work on a reciprocal JavaScript component. GPL developers consider DRM software anathema and as such an open source DRM implementation would probably not interest people willing to work with a GPL family license.

On the negative side problems with license conflicts can lead to bad reputation to the software producing organization. License conflict can affect the motivation and focus of the software development team. A reputation of misuse of licenses could lead to distrust by potential recruits and potential clients. Unmanaged license issues lead to fear, uncertainty, and doubt in the social system.

5. METHODS FOR SOFTWARE ARCHITECTURE DEVELOPMENT WITH OPEN SOURCE LICENSES

> "Enact strategy broadly, correctly and openly. Then you will come to think of things in a wide sense and, taking the void as the Way, you will see the Way as void. In the void is virtue, and no evil." [8]

This chapter covers the evaluation of known tools and methods for software engineering used to detect and resolve conflicts caused by open source license properties. Based on the interaction of open source licenses and software architecture we developed the OSSLI framework which covers the abstract legal, software architecture, procedural legal, software engineering business and social systems. The interaction between these systems is highlighted in Figure 5.1 which shows how software component integration links software architecture and open source license conflicts. Social system affects software engineering, procedural legal and business systems directly and was excluded from the figure for clarity. In order to better evaluate the mechanisms for managing these conflicts we present the the OSSLI framework in Section 5.1.. Using the OSSLI framework we evaluate license level management in Section 5.2., architecture level management in Section 5.3. and software engineering tools and methods in Section 5.4.. In the final section (Section 5.5.) we briefly evaluate the usage of the OSSLI Framework for software producing organizations beyond the license, software architecture and software engineering levels.

5.1. The OSSLI Framework

The OSSLI framework for Open Source License Management with Software Architecture is presented a series of questions covering the Abstract Legal, Software Architecture, Procedural Legal, Business, Software Engineering, Architecture, and Social Systems in Table 5.1.. If a tool or method can be used to answer these questions allows for the evaluation of how a tool or method helps manage license risk. The answers to the questions

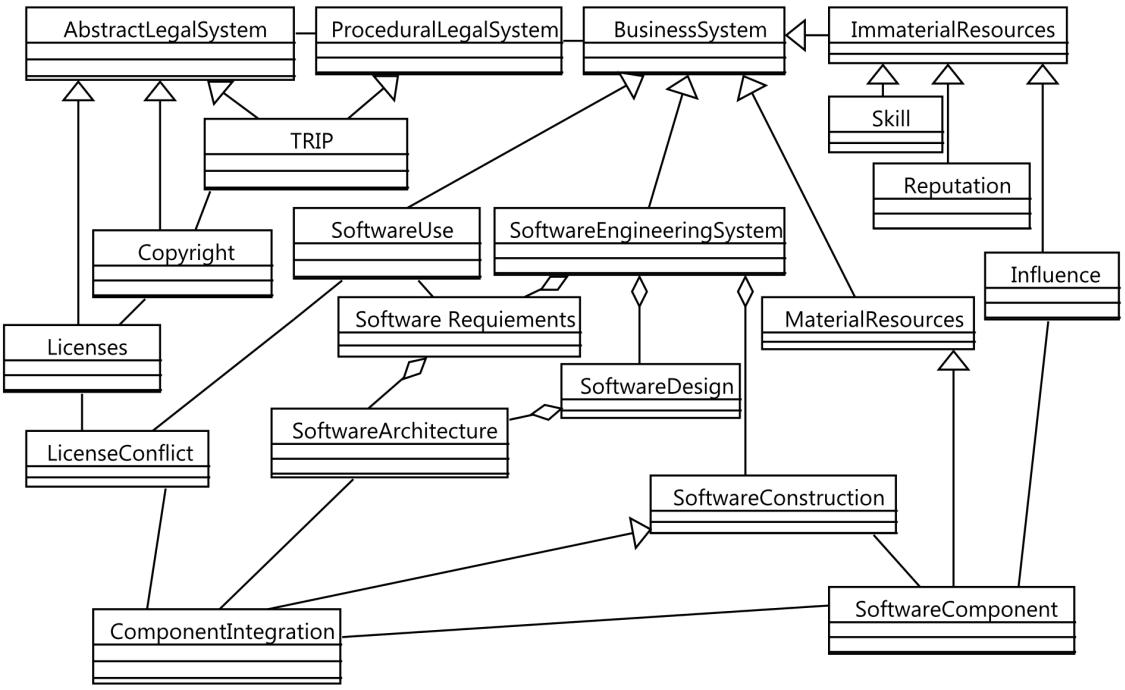

Figure 5.1: Overview of open source license effects on software development.

can also be used to evaluate the coverage of a software producing organizations license compliance methodology.

From Table 5.1. we can see that in order to evaluate the effects of open source license properties on software architecture we need only apply the questions in the abstract legal, software architecture and software engineering systems. We will use the questions in these three domains the evaluate the methods and tool used for open source license compliance in software engineering.

Table 5.1: The OSSLI framework

System	Questions
Abstract Legal (AL)	AL-0) What is the license? AL-1) Is the jurisdiction defined in the license? AL-2) What are the requirements of the license? AL-3) What are the rights granted by the license? AL-4) How are the requirements related to the right granted? AL-5) How are the licensors identified? AL-6) Can conflicts between licenses be identified?

Table 5.1: The OSSLI framework

System	Questions
	AL-7) Can conflicts between licenses be resolved?
Software Architecture (SA)	SA-0) Are the licenses of components identified? SA-1) Can the architecture be used to evaluate which licenses conflict with other license due to composition or connection? SA-2) Can the architecture be used to evaluate which licenses conflict in the different usage domains? SA-3) Can the architecture be used to resolve license conflicts? SA-4) Can the architecture be used to prevent license conflicts?
Procedural Legal (PL)	PL-0) In what jurisdiction will copyright infringement be processed? PL-1) How does the jurisdiction consider open licenses? PL-2) How does the jurisdiction consider the terms of licenses? PL-3) What are the resources needed to mange legislation related to copyright infringement? PL-4) Are potential license violations or licensors from different jurisdictions? PL-5) How will the jurisdiction take into account cultural norms?
Business (Bu)	Bu-0) Is our business model compliant with open source license requirements? Bu-1) How do open source license properties affect our intellectual capital? Bu-2) Does our use of open source components affect our business relationships? Bu-3) What resources are needed to manage open source licenses? Bu-4) What resources are needed to manage an infringement allegation or lawsuit?
Software Engineering (SE)	SE-0) Does the implementation match the software architecture? SE-1) Are all of the licenses of components identified? SE-2) Can license conflicts be detected by the software developers? SE-3) Can license conflicts be resolved by the software developers?

Table 5.1: The OSSLI framework

System	Questions
Social (So)	So-0) How will the license terms affect clients? So-1) How will the license terms affect potential recruits? So-2)How will the license terms affect the relationship with the development community? So-3 How would the software license term affect the the software engineering team ? So-4) How would infringement allegation or lawsuit customer relations? So-5) How would infringement allegation or lawsuit affect employee or potential employee relations? So-6) How would infringement allegation or lawsuit affect community relations? So-7) How would infringement allegation or lawsuit affect the software engineering team?

5.2. License Management

License modelling is not just an academic way of studying complex license texts and transforming them into more easily understandable and presentable components. Presenting licenses in structured license notation is used in industry to understand, document, and automatically process and reason about license terms, not only in conjunction with open source licenses.

We revisit the approaches of German & Hassan [30] and Alspaugh et al [35] presented in Subsection 3.1. and present two formal license modelling methods: Creative Commons Rights Expression Language (ccREL) and Open Digital Rights Language (ODRL). ccREL is developed by the the Creative Commons community and is based on RDF. ODRL is developed within the W3C and can be expressed in XML, RDF and JSON. XML (Extensible Markup Language) and JSON (JavaScript object notation) are human readable languages for representing structured data in machine readable form. RDF (Re-

source Description Framework) is an dialect of XML that allows network representation of concepts. Modelling licenses and understanding their interactions allows for the development of license compliance patterns. Patterns document how to detect certain compliance conflicts and how to resolve the conflicts.

Table 5.2: Tools for managing license on abstract legal level according to the OSSLI-framework .

	Abstract legal							
	0)	1)	2)	3)	4)	5)	6)	7)
Modelling		x	x	x	x	x		
ccREL	x		x	x		x		
ODRL	x	x	x	x	x	x		
Patterns						x	x	

Academic license models as presented German & Hassan [30] and Alspaugh et al [35] in Subsection 3.1. can be used to identify the rights and requirements of the license. The information gained by modelling licenses can be applied to wider use such as in the Qualipso project presented Section 4.3.. According to German & Hassan [30] the process of modelling license develops the modellers understanding of the license thus helping them detect license conflicts. The Qualipso project used the Carneades logic engine to automatically detect conflicts between modelled licenses based on the license properties.

ccREL is a rather simple extension of RDF. The main concepts and their relationships are presented in Figure 5.2. The model is straightforward, a work has a license and the license has properties. It is good to note that the licensor or copyright holder is identified by their relationship with the work and not the license. The ccREL model assumes that in order to receive any of the permits, the licensor must uphold all requirements and refrain from all prohibited actions, but this is not actually correct for all licenses. Such inaccuracy is can be problematic, but since the actual legal text is always going to be the primary source when evaluating more complex issue, ccREL can be useful as a lightweight notation mixing machine and human readability. It is comparatively easy to compare a representation of GPL and original BSD license in ccREL and note that original BSD has the attribution clause that does not exist in GPL and therefore conflicts with the copyleft clause. But this is not a problem if the program is not distributed so the conflict does not exist in all cases. Also the ccREL, language has only two requirements for copyleft which cover the GPL and LGPL terms of copyleft, but not Mozilla or AGPL.

The core of the ODRL model as present in Figure 5.3 shows that the complexity of

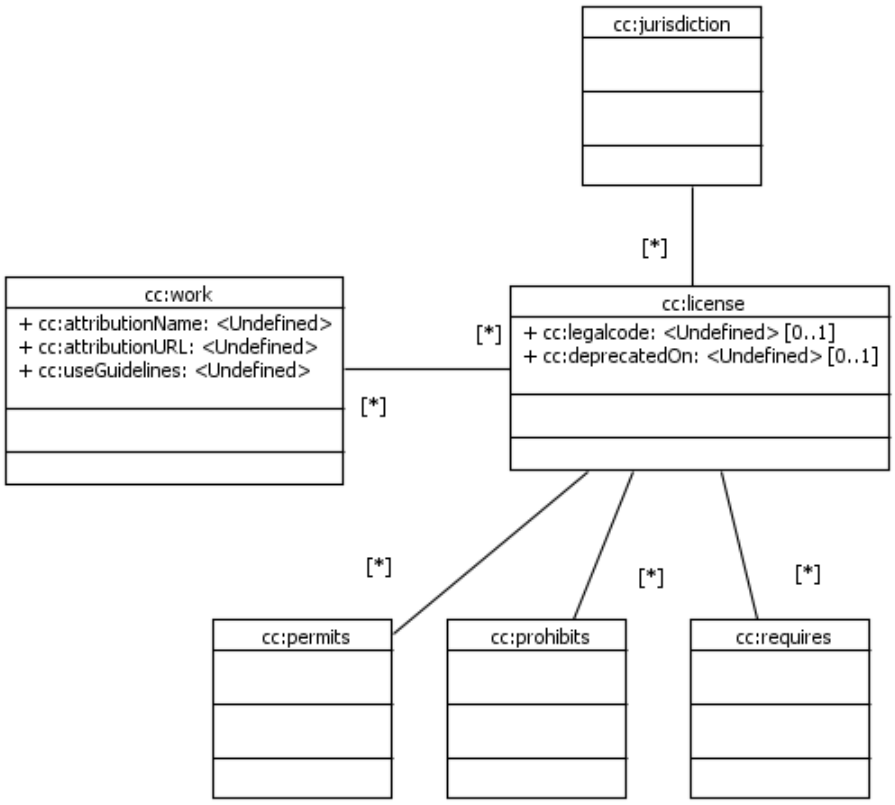

Figure 5.2: ccREL metamodel.

ODRL starts being a challenge for human readability. ODRL allows for linking requirements directly to permissions thus allowing for more detailed expression like the academic models. Gangadharan et al [46] show that ODRL can be used to model open source licenses, and that those models can be used to detect license conflicts. The expressivity and complicated nature of ODRL would make it more suitable for automated license analysis than ccREL. The core of ODRL does not take into account copyright holder relationships, but it can be extended to better describe the software development domain. Both ODRL and ccREL are suited for transferring information about licenses, storing them digitally and identifying the license and copyright holder of the work, but they should be used in conjunction with the full text of the license.

There are other license expression languages, like LicenseScript, and other legal expression languages beyond LKIF, that could be used in similar way to model open source licenses. However we found no references of them being applied to the open source domain.

According to German & Hassan [30] and Hammouda et al [48] patterns can be used

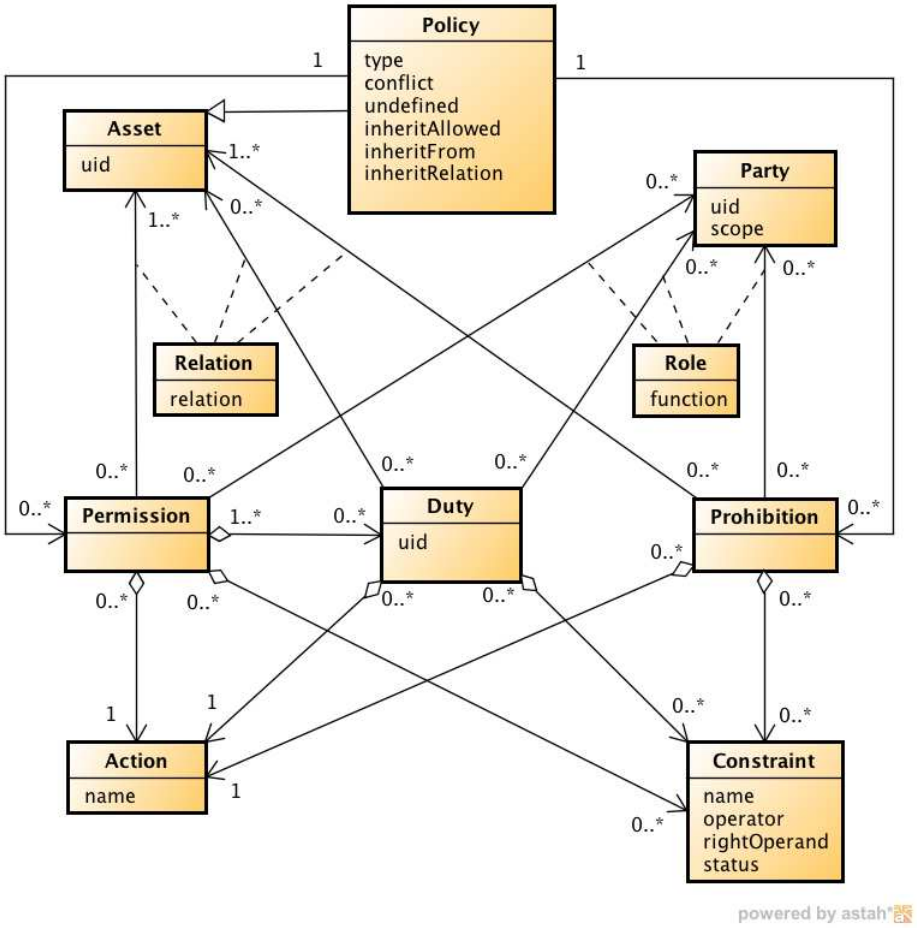

Figure 5.3: Complete version 2.0 ODRL Core Model [47].

to solve license compliance issues on the legal level. Design patterns are clearly defined solutions to common problems in a given domain. In the legal domain they are human implementable algorithms that that consist of a description of a problem and the steps required to solve the problem. The pattern presented are listed in Table 5.3. In the original sources the patterns are divided into more exact problems and more detailed solutions, as well as whether they are performed by licensor or licensee, but here they are presented in a more general sense. Licence patterns require legal domain knowledge to use, but are presented in terms that a software developer should be able to understand. The ability of the user to use a pattern to detect and resolve license compliance issues on the legal level requires that the licenses are identified and the user has suitable skill to match the license conflict to the problem description of the pattern.

Table 5.3: Patterns for the abstract legal level.

Pattern	Problem	Description	Sources
Clarification	Licence terms unclear	The licencor can issue a clarification to a norm in the license or the licensor can ask for a clarification.	[30]
Exception	Licence terms incompatible	The licensor can issue an exception to a norm in the license or the licensee can ask for a exception.	[30]
Relicensing	Licence terms incompatible	The licensor can release the software under multiple licenses, the licensee can ask for different license and some licenses allow the licensee to republish under a different license.	[30, 48]

5.3. Licenses and Software Architecture

Software architecture can be used to detect license conflicts, but it is also important to understand how the license and software architecture relate to the business requirements of the software. In the OSSLI framework we identify software architecture level issues for license compliance are repeated in Figure 5.3.

Table 5.4: The abstract legal level of the OSSLI framework

System	Questions
Abstract Legal	SA-0) Are the licenses of components identified? SA-1) Can the architecture be used to evaluate which licenses conflict with other license due to composition or connection? SA-2) Can the architecture be used to evaluate which licenses conflict in the different usage domain SA-3) Can the architecture be used to resolve license conflicts? SA-3) Can the architecture be used to prevent license conflicts?

We evaluate the benefit of deriving rules for software components used in product architecture based on license and business analysis. Other Architecture mechanism for license compliance is adding IPR annotation to software architecture descriptions and using design patterns to detect and resolve compliance issues before implementation. An overview of their functionality is presented in Table 5.5

5.3.1. License requirement for architecture

By understanding the interaction between open source licenses, software developers can define architecture level rules to prevent license conflicts. The most common approach

Table 5.5: Methods for managing license on the software legal level according to the OSSLI-framework .

	Software Architecture				
	8)	9)	10)	11)	12)
Business requirements			x		x
Consistency requirements					x
Architecture IPR annotation	x	x	x	x	x
Patterns			x	x	x

to using open source software is to limit the use of components to a selected group of licenses. The Eclipse foundation only accepts components licensed under the Eclipse Public License. A more mature approach is to analyse the business level requirements of the software and define the list suitable licenses based on those requirements while at the same time making sure the license are all compatible with each other.

There are various lists available that show which license are non-compliant with each other. By choosing a primary license for a project it is easy to compile a list of which license are compliant with the main license, but whenever a new license is added to the list it must be checked form compliance with all previously added license as well the primary license. This leads to an increasing workload for each added license.

5.3.2. Architecture decision for License Management

Adding software license information to software architecture description allows more mature evaluation of possible licenses conflicts, but this dependent on skills of the architect. This approach helps detect possible conflicts between license in by providing a visualization of their interactions. The Qualipso project uses OWL to model the architecture and IPR information [36]. Alspaugh et al [35] used Arch studio software which uses xADL architecture description language. UML is the most commonly used language to formally describe software architecture, and it can be extended by profiles to include IPR information like license and copyright holder.

Similarly to the legal domain, patterns can be used to identify and resolve license compliance conflict in the architecture system. An overview of identified patterns are given in Table 5.6. The patterns are described in more detail in the sources.

Using pattern requires some expertise in the legal as well as the software engineering domains. Most patterns require that a license incompatibility is already detected. The

Table 5.6: Patterns for the software architecture legal level.

Pattern	Problem	Description	Sources
User integration	Composition conflict	The conflicting components are distributed separately and the user integrates them as patch or plugin.	[30, 48]
Change interaction	Interaction Conflict	The type of interaction between components changed to a more loosely coupled level.	[48]
Offer as service	Composition conflict	Only one component of the conflict is distributed the other functionality is offered as service.	[48]
Tier component	Interaction Conflict	A component with a compliant license regarding to the original conflicting components is used to integrate the components.	[30, 48]

pattern helps detect the how the license mismatch can be found in the architecture. Using the patterns to resolve license issues can lead to difficulties in other areas of the program such as usability, maintainability or performance. It is important to be able to evaluate such trade-offs holistically in order to justify inclusion of the conflicting components.

5.4. License Management in Software Engineering

In a software engineering project the most likely tools to be used for license compliance are on the abstract legal, software architecture and software engineering levels. The abstract legal, software architecture and software engineering levels architecture levels are revisited in Table 5.7. Fist we present the software engineering methods for license compliance in Subsection 5.4.1. Those methods are presented int the context of the OSSLI framework in Table 5.8. Software engineering tools for license compliance are reviewed in Subsection 5.4.2. and presented in the context of the OSSLI framework in Table 5.8.

5.4.1. Methods

We identified three methods used to ensure open source license compliance in software engineering. These were educating the developers on the properties of open source licenses and the risk involved, reviewing open source source code for IPR information, and the application of legal and software architecture patterns during the software engineering process.

Table 5.7: The abstract legal, software architecture and software engineering levels of th OSSLI framework

System	Questions
Abstract Legal (AL)	AL-0) What is the license? AL-1) Is the jurisdiction defined in the license? AL-2) What are the requirements of the license? AL-3) What are the rights granted by the license? AL-4) How are the requirements related to the right granted? AL-5) How are the licensors identified? AL-6) Can conflicts between licenses be identified? AL-7) Can conflicts between licenses be resolved?
Software Architecture (SA)	SA-0) Are the licenses of components identified? SA-1) Can the architecture be used to evaluate which licenses conflict with other license due to composition or connection? SA-2) Can the architecture be used to evaluate which licenses conflict in the different usage domains? SA-3) Can the architecture be used to resolve license conflicts? SA-4) Can the architecture be used to prevent license conflicts?
Software Engineering (SE)	SE-0) Does the implementation match the software architecture? SE-1) Are all of the licenses of components identified? SE-2) Can license conflicts be detected by the software developers? SE-3) Can license conflicts be resolved by the software developers?

The most versatile method to manage risk on open source licenses is **education** of developers on licensing issues. Being aware of license conflict possibilities and repercussions will help developers stay aware of license terms. Being able to identify licenses forms the basis of being able to detect license conflicts and resolve them. Our review did not find any specific methods or metrics for education on opens source license. The Linux foundation has developed a guide for implementing an open source license compliance program for a software producing organization, which could be used as a basis of what a licenses compliance education program could contain.

Package review is the process of reviewing the source code of software packages and documenting all license texts and copyright notices found within. This is very useful as not all open source components are licensed with the same license that they are distributed with. If the package contains licenses tat conflict with each or the distribution license it can not be used without copyright infringement. It is possible to infringe on an undocumented license that is contained in the component. Von Willebrand & Partanen [49] consider package review as a necessary part of any license compliance program. The result of the audit can be included into the software architecture.

Table 5.8: Methods for managing license on software engineering level according to the OSSLI-framework.

	Abstract legal								Software architecture					Software engineering			
	0)	1)	2)	3)	4)	5)	6)	7)	0)	1)	2)	3)	4)	0)	1)	2)	3)
Review	x					x	x			x				x	x		
Education	x	x	x	x	x	x	x	x	x	x	x	x	x		x	x	x
Patterns						x	x	x				x	x		x	x	

Table 5.9: Tools for managing license on software engineering level according to the OSSLI-framework .

	Abstract legal							Software architecture						Software engineering			
	0)	1)	2)	3)	4)	5)	6)	0)	1)	2)	3)	4)	5)	0)	1)	2)	3)
Antepedia	x																
ASLA	x						x	x	x					x	x	x	
DCT	x						x	x	x					x	x	x	
Fossology	x					x											
HUT OSLC	x					x	x								x	x	
Lchecker	x							x	x						x		
Ninka	x														x		
Qualipso Carneades			x	x	x	x		x									
Qualipso OSLC	x														x		
OSSLI-tool	x	x	x	x	x	x	x	x	x	x	x	x	x		x	x	x

Software architecture and abstract legal and software architecture license conflict resolution **patterns** have already been covered in their own corresponding sections, but they can also be used during the software engineering process. In order to properly implement a patterns in practice software engineering, legal or social skills may be required.

5.4.2. Tools

We attempted to find a representative sample of open source license compliance tools. We preferred programs that were open source or documented in peer reviewed sources. While the sample size is small it represents the mainstream of approaches these issues and the resources needed to map every program that can be used to detect a license text or extend a software architecture description with IPR information were unavailable.

Antepedia is a web service which can be used to search for license documentation of open source components programs, such as license and copyright information (AL-0). It includes information not only of published license but also licenses of internal components. [50]

ASLA (Automated Software License Analyzer) is developed by Realtor and it uses static analysis to detect license and copyright information of files (AL-0, SA-0, SE-0, SE-1). ASLA reverse engineers the program architecture based on compiler output to detect dynamic and static links between licenses and analyze the composition for possible license conflict based on a predefined ruleset (SA-1, SE-2). [51]

The **DCT** (Dependency Checker Tool) developed by the Linux Foundation analyses links between binaries and uses license information from package manager to detect potentially conflicting linkage types between license (AL-0, SA-0, SA-1, SE-0, SE-1, SE-2). The rules for license and link conflicts are developed by the Linux Foundation (Al-6). [52]

FOSSology searches files and packages for copyright and license information. It can be used to build a database of file and package copyright information using a notation such as SPDX or another defined schema (AL-0, AL-5). It is an open source project that has many large industrial users.[53]

HUT OSLC is a source code analysis tool, and there are a couple of different versions available of this Open source License Checker that were developed at Helsinki University of Technology, whose properties vary a bit. The HUT OSLC identifies license and copy-

right information from static analysis of source files(AL-0, AL-5, SA-0, SE-1). It uses a predefined compatibility table of licenses to detect license conflicts(AL-6, SE-2).[54]

Lchecker detects open source code in source code by matching it to Google code search(SE-1). If matches are found it provides the license based on information in Google code search(AL-0).[55]

Ninka detects licenses and copied code from source code files (AL-0, SE-1). Ninka needs an open source repository to match the code to. Code duplicates resembling published open source are reported. [56]

Qualipso Carneades is used to reason about license conflicts based in a software architecture based on license models(AL-2, AL-3, AL-4). The license models are described in LKIF-OWL and the architecture descriptions are transformed from UML to OWL (SA-0, SA-1). These OWL models processed by the Carneades logic engine using a compliance ruleset described in OWL to detect conflicts (SE-2).[36]

Qualipso OSLC detects licenses from source code files and sends them to a web service for conflict checking (SA-0, SE-0). The conflict detections mechanism used at the web service are not described (SE-2).[57]

The **OSSLI tool** is mentioned here for completeness and comparison. The OSSLI-tool was developed with the help of the OSSLI Framework is covered in detail Chapter 6.

Binary Analysis Tool developed by the Linux foundation is used to by license compliance investigators to detect open source licensed code in compiled software packages by similarity metrics. This is not really an architecture or license property tool, but it is mentioned since it is the primary tool, in addition to tip-offs, for license compliance investigators.

There a numerous companies that provide proprietary tools an services for open source license compliance. Due to their proprietary nature they are not reviewed in the scope of this thesis.

5.4.3. Review

While there are other tools available, based on our review there are five main functions for compliance tools: detecting code and licenses, identifying IPR information, analyzing the composition, detecting license conflicts and resolving conflicts. There is clear dif-

ferentiation between analysis tools and design tools and only the OSSLI-tool supported conflict resolution. This differentiation can be seen in Table 5.10 In order to ensure license compliance a mixture of tools and methods should be used.

Table 5.10: Comparing roles license compliance tools.

Tool	Detection	Identification	Analyzing	Conflict	Resolution
Asla	yes	yes	yes	yes	no
Fossology	yes	yes	no	no	no
HUT OSLC	yes	yes	no	yes	no
LChecker	yes	yes	no	no	no
Qaalipso Carneades	no	no	yes	yes	no
OSSLI	no	no	yes	yes	yes

5.5. License Management in Software Production

In order for an organization to benefit from using open source and be free of the fear of license non-compliance in software development legal, architecture and software engineering tools are not enough. These tools and methods must be supported from procedural legal, business and social levels.

5.5.1. Legal Proceedings

It is suggested that software producing organizations prepare themselves for the risks involved in IPR litigation. The resources needed for proceedings include legal skills and official representation, but its is also necessary to understand legal customs. The definition of due diligence may vary by jurisdiction and it is necessary to be able to show reasonable attempt at license compliance. This will require both software engineering and legal expertise and organization should be aware of where it could get these resources if faced with litigation. Including architecture level methods of license compliance will in many cases support the claim of due diligence. The definition of reasonable will vary by jurisdiction as no definition of due diligence has been recorded for open source license compliance.

5.5.2. License Management in Business

Education on open source licenses on a business level is useful, since by choosing a suitable business model for the software products usage domain many license conflicts can be avoided. Some of the resolution mechanism, like purchasing a proprietary license for an open source product or hosting a part of the product as a service will affect the software business functionality. Education on licensing issues also allows risk evaluation by the business level of the organization and their vigilance on the issues will support the developers attention to license compliance.

Ziemr [42] suggests that clearly differentiating and recording proprietary and non proprietary resources on a business level helps in preventing license conflicts in software development. By having a clear separation, accidentally mixing open and closed source is prevented.

Managing open source licensing can create other benefits such as resource of developer skilled with working with same tools as the organization or an increased supply of complimentary businesses. Using and contributing to open source can offer other benefits such as increase social capital.

5.5.3. OS-communities and Social Effects of Licenses

Maintaining a relationship with open source communities will allow a software producing organization to develop their understanding of license interpretation customs. Positive community relations will also help in recruitment efforts. A positive reputation among the development community will allow greater leverage in influencing open source projects development. This leverage cannot be used without complying with and understanding licenses and the community.

6. OSSLI TOOL

> "Once men turned their thinking over to machines in the hope that this would set them free. But that only permitted other men with machines to enslave them." [37]

In the course of the OSSLI project the OSSLI tool was developed in tandem with our research. The OSSLI tool reflects the understanding of open source license property and software architecture interaction that was formalized into the OSSLI framework. The OSSLI tool is based on extending software architecture descriptions with IPR information which can then be used to manage license issues before or during software development. The tool is developed as an Eclipse plugin for the Papyrus UML editor. The OSSLI tool and the UML profiles are documented in detail by Luoto [58].

6.1. Tool Design

The OSSLI tool is based on a open architecture in which the core communicates with UML model in Papyrus. There are nine types of plugin that implement the actual license compliance support: Conflict Detection, Problem resolution, Package Database, Risk View, License Model, Logger, Reporting, Profile and Help. The plugin architecture is shown Figure 6.1 and the functions of the plugins are described in Table 6.1

Table 6.2 restates the the OSSLI-framework for the abstract legal, software architecture and software engineering levels. Table 6.3 shows how the roles of the different plugins map to the OSSLI framework.

The OSSLI tool is mainly a framework for showing how to integrate different approaches to open source license compliance into one tool. For example we implemented both package review based profile and ccREL based profile. The package review based profile was used by different plugins to reason whether a package is safe for certain usage domain. The ccREL was profile could be used by different plugins to reason about

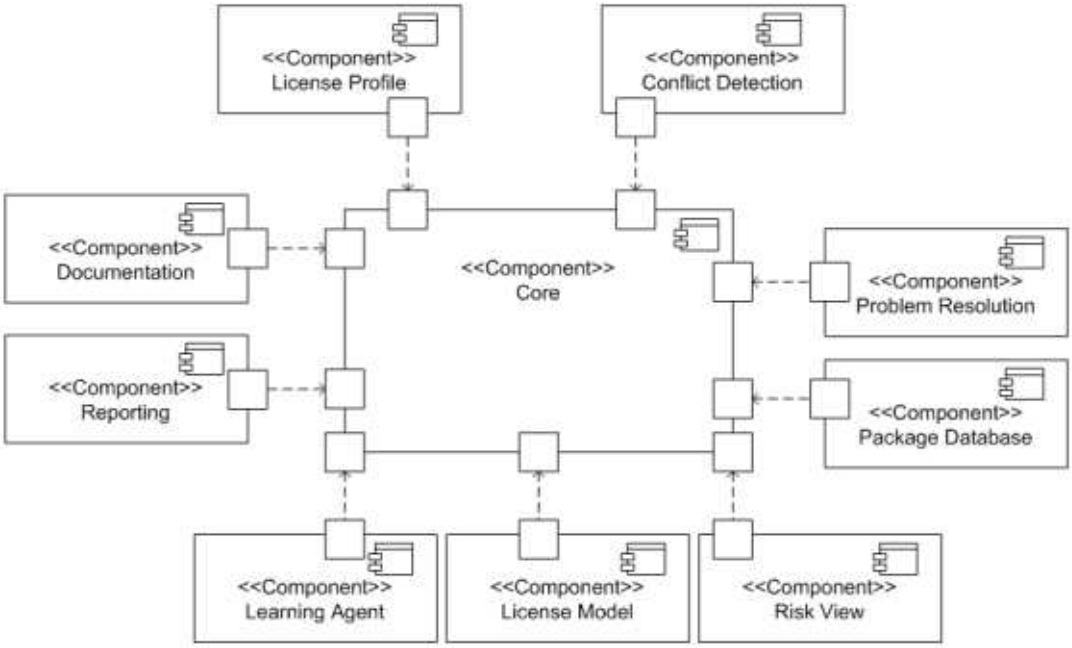

Figure 6.1: The architecture of the OSSLI tool.

copyleft conflicts based on linking or composition of components. A plugin could be developed combining the information from these different plugins to support even more refined reasoning on license risks.

The OSSLI tool represents the holistic approach to license compliance advocated by the OSSLI framework. It is a tool to be used during software requirements and architecture engineering. As such it lacks the source code level license and copyright information procurement tools. If the Papyrus tools architecture reverse engineering tools are at some point mature enough they could be extended to include the IPR detection functionality found in other license compliance tools. The nature of reverse engineering tools is that they are applied after the fact, whereas the OSSLI framework and OSSLI tool are about preemptive action to ensure license compliance. Therefore no plugin was developed to OSSLI framework question SE-0.

6.2. CCREL and Copyleft Management

Based on the ccREL license models plugins where developed that could be used to detect cofnlicts with LGPL or GPL and other licenses. All the plugins named here are in the org.eclipse.papyrus.ossli namespace. The ccrelprofile plugin adds IPR information based

Table 6.1: Description of the functions of the OSSLI plugins according to Luoto[58].

Component	Description
Core	Handles interactions between the application model, licensing information and the user.
License Profile	An UML extension to include license information.
License Model	It describes in computable format the clauses, restrictions, rights and the interdependencies of a license.
Package Database	A repository containing a list of packages with license, copyright and other IPR related information
Risk View	Assess legal risks related to use of component for variable purposes re-licensing, sale, internal use etc.
Conflict Detection	Analysis whether license terms of different licenses conflict when linked or interconnected with another way into the same software.
Problem Resolution	Suggests operations that can be performed to remove license conflicts from model.
Learning Agent	Records user actions so that they can be later used to improve program performance.
Reporting	The analysis results from the different components can be output in different formats.
Documentation	Provides a way to linking to internal and external documentation on open source licensing concerns.

on the ccREL standard to UML-models, it also adds a possibility to define associations between components in terms relevant to copyleft compliance. ccrelicenses is a license model plugin that expresses ccREL license models in java. Ccrelcopyleftconflicts is a conflict detection plugin that associates the can perform analysis based on the UML-model based on the information in the profiled UML-model and the license.

The algorithm of the Ccrelcopyleftconflicts plugin is based on the OSSLI framework. Based on the terms of the GPL and LGPL copyleft requirements we defined rules for Copyleft and Lesser Copyleft range based on customs as defined in the "Linking Document." [59] For both Lesser Copyleft and Copyleft components in the same compilation unit or package will be in range of copyleft. Also a component linked statically to a component within copyleft range will be in copyleft range for both Copyleft and Lesser Copyleft, but dynamically linked components will be in copyleft range of only Copyleft but not Lesser Coptyleft. For each component with Copyleft or Lesser Copylef requirement in their license a full spannign tree for copylef range is calculated. For each component in copyleft range the licence model is checked for requirements that do not appear in the original license. If such requirement are found then then a copyleft conflict is found

Table 6.2: The abstract legal, software architecture and software engineering levels of th OSSLI framework

Abstract Legal	Software Architecture	Software Engineering
AL-0) What is the license? AL-1) Is the jurisdiction defined in the license? AL-2) What are the requirements of the license? AL-3) What are the rights granted by the license? AL-4) How are the requirements related to the right granted? AL-5) How are the licensors identified? AL-6) Can conflicts between licenses be identified? AL-7) Can conflicts between licenses be resolved?	SA-0) Are the licenses of components identified? SA-1) Can the architecture be used to evaluate which licenses conflict with other license due to composition or connection? SA-2) Can the architecture be used to evaluate which licenses conflict in the different usage domains? SA-3) Can the architecture be used to resolve license conflicts? SA-4) Can the architecture be used to prevent license conflicts?	SE-0) Does the implementation match the software architecture? SE-1) Are all of the licenses of components identified? SE-2) Can license conflicts be detected by the software developers? SE-3) Can license conflicts be resolved by the software developers?

Table 6.3: The functions of the OSSLI-tool plugins mapped to the OSSLI framework.

	Abstract legal								Software architecture					Software engineering			
	0)	1)	2)	3)	4)	5)	6)	7)	0)	1)	2)	3)	4)	0)	1)	2)	3)
License Profile	x	x				x			x						x		
License Model	x	x	x	x	x	x											
Package Database	x	x				x			x						x		
Risk View											x						
Conflict Detection							x			x		x	x			x	
Problem Resolution								x				x	x				x
Learning Agent								x									x
Reporting	x	x	x	x	x	x	x	x	x	x	x	x	x		x	x	x

and the relationship between components and conflicting licenses are added to and array which the plugin returns.

The ccrelcopyleftconflictsanalysis plugin visualizes the conflicts found by the Ccrelcopyleftconflicts plugin in a table that shows all relationships on one axis and the conflict path on another. The user can then select which component relationships should be changed to resolve the copyleft conflict or conflicts. These selected relationships and conflicting license are returned by the plugin.

The ccrelcopyleftconflictresolver plugin takes out put of the ccrelcopyleftconflictsanalysis plugin and based on the type of conflict and relationship suggests actions that can be taken to resolve the conflict. These actions are based on the open source compliance patterns of German & Hassan [30]] and Hammouda et al [48]. Possible options include moving a conflicting component out of the compilation package and changing the type of the relationship between components.

The creation of the ccREL copyleft management process is a simple constructive application of the OSSLI framework. While the tools use requires an understanding of both legal and software aspects of open source the tools, it shows the value of the OSSLI framework by bringing abstract legal, procedural legal and software architecture concerns together.

7. DISCUSSION AND EVALUATION

"Do nothing which is of no use." [8]

In this chapter we evaluate the methodologies used in the research and as well as the whether of the application of the OSSLI framework to evaluate license compliance tools and methods or develop the OSSLI tool provided meaningful information on the accuracy or usefulness off the OSSLI framework. We also consider the practical applications of the results of this research.

7.1. Methodology

There were numerous problems with the implementation of the methodologies chosen. But the choice of theoretical basis in General System Theory and ontologies provided a well rounded approach to the problem. The General System Theory and Sowa's KRO mesh well, and provide a suitably abstract point of view that helped clarify a lot of the complexity. In particular merging the ontologies was difficult. LKIF and KRO were loosely at the same abstraction level, so there were some inconsistencies due to the different approaches. The software engineering ontology was not rigorous in its subclass definitions, many of which should have been "is a part of" relationships instead. This made merging it with other ontologies difficult.

Literature review was problematic. Our database selection criteria were flawed. The different search options between databases made inputting the searches problematic but the quantity of results allays fears that these problems would affect the validity of the material gathered. Because the ACM and IEEE work together in with their libraries, all results found in the ACM digital library were duplicates. In the future only one of these databases should be chosen.

There were some problems with license selection. We realized that the citation counts given by the different databases were not comparable so we chose to gather them inde-

pendently instead using Google Scholar. Google Scholar does not differentiate between chapter and book citations, so the citation counts between book chapters and journal articles or conference papers were not comparable. This was resolved by disregarding all book chapter from the literature review results. The final and biggest problem was the low quality abstracts of software engineering papers in general. An abstract should present research methods and findings, but many papers lacked either one or both leaving only the subject of research in the abstract. If relevant articles were disregarded from the literature review based on the poor quality of their abstracts maybe their content would have been similar quality, so the loss is not great.

7.2. OSSLI Framework

The OSSLI framework on its own works as a check list for developing open source license compliance on a more nuanced level than most approaches. It is based on a solid theoretical framework that connects the gaps between previous research. The evaluation of tools and methods in relation to the OSSLI framework shows that a majority of tools only serve small function and that holistic approaches like educating developers on licensing issues and lawyers on software engineering as well as understanding open source are necessary for licenses.

The OSSLI framework shows how complex the issues that lead to license conflicts with open source license are. Due to this complexity automated analysis tools as well as visualization tools for architecture and licenses are necessary.

7.3. Usefulness of Findings

Based on the evaluation of the OSSLI tool it could be used to support open source license compliance in a software engineering organization, but practical use experience and feedback would be needed to better evaluate whether such use is actually practical.

Based on the tools and methods evaluation tools could be grouped so that an organization could choose the best tool for each function based on their particular situation. The evaluation results could also be used to ensure that the organizations license compliance process uses a suitably varied approach in order to comprehensively guard against inadvertent copyright infringement.

All findings in this research have been qualitative in nature and as such may not be practically applicable in every situation. The framework does provide a basis for further quantitative research in the subject.

8. CONCLUSIONS

> "I must not fear. Fear is the mind-killer. Fear is the little-death that brings total obliteration. I will face my fear. I will permit it to pass over me and through me. And when it has gone past I will turn the inner eye to see its path. Where the fear has gone there will be nothing. Only I will remain." [37]

In conclusion it is necessary to focus on what has been learned by developing the OSSLI framework and what applications this research suggest for the future.

Licence conflict risk is a complex issue but it can be managed. The same properties which make open source license catalysts for software development also lead to the risk of compliance issues. Open source development practices also increase the risk of unidentified components in source code.

The OSSLI framework helps ensure that license compliance is covered from all the necessary angles and not just by legal and software engineering methods. The research also shows that architecture level evaluations of IPR issues can be used to reduce the risk.

In order to properly evaluate the costs and benefits of managing open source license the integration of tools and methods to the software engineering process needs to be researched and evaluated in practice. This means usability research and quantitative research on tool and method efficiency.

On the other hand the benefits open source use need to researched quantitatively in order to contrast them with the results of quantitative research on license risk that is also needed. There are limited opportunity to research license non-compliance and it effects, due to a lack of legal precedents. Research on applying software copyright internationally would also help to evaluate the risk.

Fear of the unknown is also fear of the open source community. Research on effects of community integration on the risks and benefits of using open source is needed, both

quantitative and qualitative studies would help. Showing the benefits and methods of community integrations, and the community integration itself, would clearly help a lot when facing the uncertainty related to open source licenses. Holistic research covering license properties and software engineering process and software business would help place this research in a context to bring more clarity to those who still fear the open source licenses.

BIBLIOGRAPHY

[1] George Lucas. Star Wars: Episode I - The Phantom Menace, 1999.

[2] Eric Raymond. The cathedral and the bazaar. *Knowledge, Technology & Policy*, 12(3):23–49, 1999.

[3] Systems and software engineering – architecture description. *ISO/IEC/IEEE 42010:2011(E) (Revision of ISO/IEC 42010:2007 and IEEE Std 1471-2000)*, pages 1–46, 2011.

[4] E.S. Raymond. Why Microsoft smears-and fears-open source. *Spectrum, IEEE*, 38(8):14–15, 2001.

[5] John F. Sowa. *Knowledge Representation: Logical, Philosophical and Computational Foundations*. Brooks/Cole, 2000.

[6] R. Hoekstra, J. Breuker, M. Di Bello, and A Boer. The LKIF Core Ontology of Basic Legal Concepts. In *Proceedings of the Workshop on Legal Ontologies and Artificial Intelligence Techniques*, 2007.

[7] Pornpit Wongthongtham, Elizabeth Chang, Tharam Dillon, and Ian Sommerville. Development of a Software Engineering Ontology for Multisite Software Development. *IEEE Trans. on Knowl. and Data Eng.*, 21(8):1205–1217, August 2009.

[8] Musashi Miyamoto. *Go Rin No Sho: A Book of Five Rings*. 1645.

[9] IEEE Systems and software engineering – Vocabulary. *ISO/IEC/IEEE 24765-2010(E)*, pages 1 – 418, 2010.

[10] Samuel A. Ajila and Di Wu. Empirical study of the effects of open source adoption on software development economics. *Journal of Systems and Software*, 80(9):1517 – 1529, 2007.

[11] Diomidis Spinellis and Clemens Szyperski. How is open source affecting software development? *IEEE Software*, 21(1):28–33, 2004.

[12] Weibing Chen, Jingyue Li, Jianqiang Ma, Reidar Conradi, Junzhong Ji, and Chunnian Liu. An empirical study on software development with open source components in the chinese software industry. *Software Process: Improvement and Practice*, 13(1):89–100, 2008.

[13] C. Ruffin and C. Ebert. Using open source software in product development: a primer. *Software, IEEE*, 21(1):82–86, 2004.

[14] Christof Ebert. Open Source Drives Innovation. *Software, IEEE*, 24(3):105–109, 2007.

[15] D. Bahn and D. Dressel. Liability and Control Risks with Open Source Software. In *Information Technology: Research and Education, 2006. ITRE '06. International Conference on*, pages 242–245, 2006.

[16] David McGowan. Legal implications of open-source software. *U. Ill. L. Rev.*, page 241, 2001.

[17] Lawrence Rosen. *Open Source Licensing: Software Freedom and Intellectual Property Law*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2004.

[18] Bruce Perens et al. The open source definition. *Open sources: voices from the open source revolution*, pages 171–85, 1999.

[19] Andrew Sinclair. Licence profile: Apache license, version 2.0. *International Free and Open Source Software Law Review*, 2(2), 2011.

[20] Alain Abran, Pierre Bourque, Robert Dupuis, James W. Moore, and Leonard L. Tripp. *Guide to the Software Engineering Body of Knowledge - SWEBOK*. IEEE Press, Piscataway, NJ, USA, 2004 version edition, 2004.

[21] M. Fowler. Design - Who needs an architect? *Software, IEEE*, 20(5):11–13, 2003.

[22] David Garlan. Software architecture: a roadmap. In *Proceedings of the Conference on The Future of Software Engineering*, ICSE '00, pages 91–101, New York, NY, USA, 2000. ACM.

[23] Donald Hislop. *Knowledge management in organizations: a critical introduction*. Oxford University Press, USA, 2005.

[24] Pearl Brereton, Barbara A. Kitchenham, David Budgen, Mark Turner, and Mohamed Khalil. Lessons from applying the systematic literature review process within the software engineering domain. *J. Syst. Softw.*, 80(4):571–583, April 2007.

[25] Lars Skyttner. *General Systems Theory: Ideas & Applications*. World Scientific Publishing Co. Pte. Ltd., Singapore, 2001.

[26] Ludvig von Bertalanffy. *General System Theory: Foundations, Development, Applications*. George Braziller, Inc., New York, third printing edition, 1968.

[27] Thomas R. Gruber. A translation approach to portable ontology specifications. *Knowlegde Acquisition*, 5(2):199–220, June 1993.

[28] Natalya Fridman Noy and Carole D. Hafner. The state of the art in ontology design: A survey and comparative review. *AI Magazine*, 18(3):53–74, 1997.

[29] John F. Sowa. Ontology. `http://www.jfsowa.com/ontology/`, 2010. Accessed: 17.4.2013.

[30] D.M. German and A.E. Hassan. License integration patterns: Addressing license mismatches in component-based development. In *Software Engineering, 2009. ICSE 2009. IEEE 31st International Conference on*, pages 188–198, 2009.

[31] F.P. Gomez and K.S. Quinones. Legal Issues Concerning Composite Software. In *Composition-Based Software Systems, 2008. ICCBSS 2008. Seventh International Conference on*, pages 204–214, 2008.

[32] Mikko Välimäki. *Oikeudet tietokoneohjelmistoihin*. Talentum. Helsinki, 2009.

[33] Malcolm Bain. Software Interactions and the GNU General Public License. *International Free and Open Source Software Law Review*, 2(2):165–179, 2010.

[34] C. Forbes, I. Keivanloo, and J. Rilling. When open source turns cold on innovation - the challenges of navigating licensing complexities in new research domains. In

Software Engineering (ICSE), 2012 34th International Conference on, pages 1447–1448, 2012.

[35] Thomas A Alspaugh, Hazeline U Asuncion, and Walt Scacchi. Intellectual property rights requirements for heterogeneously-licensed systems. In *Requirements Engineering Conference, 2009. RE'09. 17th IEEE International*, pages 24–33. IEEE, 2009.

[36] Thomas F. Gordon. Analyzing Open Source License Compatibility Issues with Carneades. In *Proceedings of the 13th International Conference on Artificial Intelligence and Law*, ICAIL '11, pages 51–55, New York, NY, USA, 2011. ACM.

[37] Frank Herbert. *Dune*. Chilton Books, USA, 1965.

[38] Elspeth Berry, Matthew J Homewood, and Barbara Bogusz. *Complete EU Law: Text, Cases, and Materials*. Oxford University Press, 2013.

[39] Christopher May. Escaping the TRIPs' trap: The political economy of free and open source software in Africa. *Political Studies*, 54(1):123–146, 2006.

[40] Jyh-An Lee. Not to profit from open source: The role of nonprofit organizations in open source software development. In *Professional Communication Conference, 2008. IPCC 2008. IEEE International*, pages 1–9, July 2008.

[41] Pekka Himanen. *The hacker ethic*. Random House, 2001.

[42] S. Ziemer. On the adoption of open source software in aeronautics. In *Aerospace Conference, 2012 IEEE*, pages 1–10, 2012.

[43] J. Lindman, A. Paajanen, and M. Rossi. Choosing an open source software license in commercial context: A managerial perspective. In *Software Engineering and Advanced Applications (SEAA), 2010 36th EUROMICRO Conference on*, pages 237–244, 2010.

[44] Bhasker Mukerji, Bharat Maheshwari, Vinod Kumar, and Uma Kumar. A review of dominant open source software business models. In *ASAC*, volume 29, 2008.

[45] Carlo Daffara. Business models in floss-based companies. In *Workshop presentatioon at the 3rd Conference on Open Source Systems (OSS 2007)*, 2007.

[46] G.R. Gangadharan, Vincenzo DÁndrea, Stefano Paoli, and Michael Weiss. Managing license compliance in free and open source software development. *Information Systems Frontiers*, 14(2):143–154, 2012.

[47] Renato Iannella, Susanne Guth, Daniel Pähler, and Andreas Kasten. ODRL version 2.0 core model. Specification. `http://www.w3.org/community/odrl/two/model/`, Apr 2012. Accessed: 4.5.2014.

[48] Imed Hammouda, Tommi Mikkonen, Ville Oksanen, and Ari Jaaksi. Open Source Legality Patterns: Architectural Design Decisions Motivated by Legal Concerns. In *Proceedings of the 14th International Academic MindTrek Conference: Envisioning Future Media Environments*, MindTrek '10, pages 207–214, New York, NY, USA, 2010. ACM.

[49] Martin von Willebrand and MikkoPekka Partanen. Package review as a part of free and open source software compliance. 2010.

[50] Antelink. Antepedia. `http://www.antepedia.com/pages/more`, 2012. Accessed: 17.5.2014.

[51] T. Tuunanen, J. Koskinen, and T. Karkkainen. Asla: reverse engineering approach for software license information retrieval. In *Software Maintenance and Reengineering, 2006. CSMR 2006. Proceedings of the 10th European Conference on*, pages 4 pp.–294, March 2006.

[52] Jeff Licquia Stew Benedict. Dependency Checker Tool: Overview and Discussion. `http://www.linuxfoundation.org/sites/main/files/publications/lf_foss_compliance_dct.pdf`. Accessed: 17.5.2014.

[53] Robert Gobeille. The fossology project. In *Proceedings of the 2008 International Working Conference on Mining Software Repositories*, MSR '08, pages 47–50, New York, NY, USA, 2008. ACM.

[54] Yuan Yuan Mika Rajanen Xie Xiaolei Lauri Koponen Veli-Jussi Raitila Jussi Sipoma Jing Jing-Helles, Sakari Kääriäinen. Open Source License Checker 2.0 ReadMe. `http://www.soberit.hut.fi/T-76.4115/06-07/projects/groups/14/i2/readme.html`, 2007. Accessed: 17.5.2014.

[55] Hongyu Zhang, Bei Shi, and Lu Zhang. Automatic checking of license compliance. In *Software Maintenance (ICSM), 2010 IEEE International Conference on*, pages 1–3, Sept 2010.

[56] Daniel M. German, Yuki Manabe, and Katsuro Inoue. A sentence-matching method for automatic license identification of source code files. In *Proceedings of the IEEE/ACM International Conference on Automated Software Engineering*, ASE '10, pages 437–446, New York, NY, USA, 2010. ACM.

[57] HongBo Xu, HuiHui Yang, Dan Wan, and Jiangping Wan. The design and implement of open source license tracking system. In *Computational Intelligence and Software Engineering (CiSE), 2010 International Conference on*, pages 1–4, Dec 2010.

[58] Antti Luoto. A UML Profile Approach to Managing Open Source Software Licensing. Masters thesis, Tampere University of Technology, 2013.

[59] European Legal Network. Working Paper on the legal implication of certain forms of Software Interactions (a.k.a linking).

A. REVIEWED ARTICLES

Reference Type	Authors, Primary	Title Primary	Periodical Full	Pub Year	Vol-ume	Issue	Cited
Journal Article	Graham,L.	Legal implications of operating systems	Software, IEEE	1999	16	1	6
Journal Article	Coleman,E. Gabriella	High-Tech Guilds in the Era of Global Capital	Anthropology of Work Review	2001	22	1	12
Journal Article	Ming-Wei Wu; Ying-Dar Lin	Open source software development: an overview	Computer	2001	34	6	84
Journal Article	Gallivan,Michael J.	Striking a balance between trust and control in a virtual organization: a content analysis of open source software case studies	Information Systems Journal	2001	11	4	220
Journal Article	Johnson,Justin Pappas	Open Source Software: Private Provision of a Public Good	Journal of Economics & Management Strategy	2002	11	4	219
Conference Proceedings	Egyedi,T. M.; van Wendel de Joode,R.	Standards and coordination in open source software	Standardization and Innovation in Information Technology, 2003. The 3rd Conference on	2003			7
Journal Article	Ruffin,C.; Ebert,C.	Using open source software in product development: a primer	Software, IEEE	2004	21	1	78
Journal Article	GRAHAM,STUART J. H.; MOWERY,DAVID C.	The Use of USPTO ?Continuation? Applications in the Patenting of Software: Implications for Free and Open Source*	Law & Policy	2005	27	1	10
Conference Proceedings	Stewart,K. J.; Ammeter,A. P.;Maruping,L. M.	A Preliminary Analysis of the Influences of Licensing and Organizational Sponsorship on Success in Open Source Projects	System Sciences, 2005. HICSS '05. Proceedings of the 38th Annual Hawaii International Conference on	2005			25
Conference Proceedings	Parker,G.; Van Alstyne,M.	Mechanism Design to Promote Free Market and Open Source Software Innovation	System Sciences, 2005. HICSS '05. Proceedings of the 38th Annual Hawaii International Conference on	2005			12
Journal Article	MacCormack,Alan; Rusnak,John; Baldwin,Carliss Y.	Exploring the Structure of Complex Software Designs: An Empirical Study of Open Source and Proprietary Code	Management Science	2006	52	7	384
Journal Article	Bonaccorsi,Andrea; Giannangeli,Silvia; Rossi,Cristina	Entry Strategies under Competing Standards: Hybrid Business Models in the Open Source Software Industry	Management Science	2006	52	7	245
Journal Article	Stewart,Katherine J.; Gosain,Sanjay	The Impact of Ideology on Effectiveness in Open Source Software Development Teams	MIS Quarterly	2006	30	2	300
Journal Article	Cook,Ian; Horobin,Gavin	Implementing eGovernment without promoting dependence: open source software in developing countries in Southeast Asia	Public Administration and Development	2006	26	4	12

Reference Type	Authors, Primary	Title Primary	Periodical Full	Pub Year	Volume	Issue	Cited
Conference Proceedings	Turner,A. J.	The development and use of open-source spacecraft simulation and control software for education and research	Space Mission Challenges for Information Technology, 2006. SMC-IT 2006. Second IEEE International Conference on	2006			4
Journal Article	Lerner,Josh; Pathak,Parag A.; Tirole,Jean	The Dynamics of Open-Source Contributors	The American Economic Review	2006	96	2	66
Journal Article	Burk,Dan L.	Intellectual Property in the Context of e-Science	Journal of Computer-Mediated Communication	2007	12	2	20
Journal Article	C?mara,Gilberto; Fonseca,Frederico	Information policies and open source software in developing countries	Journal of the American Society for Information Science and Technology	2007	58	1	70
Journal Article	Murray,Fiona; O'Mahony,Siobh?n	Exploring the Foundations of Cumulative Innovation: Implications for Organization Science	Organization Science	2007	18	6	91
Conference Proceedings	West,J.	Value Capture and Value Networks in Open Source Vendor Strategies	System Sciences, 2007. HICSS 2007. 40th Annual Hawaii International Conference on	2007			41
Conference Proceedings	Sarkinen,J.	An open source(d) controller	Telecommunications Energy Conference, 2007. INTELEC 2007. 29th International	2007			3
Journal Article	Polanski,Arnold	Is the General Public Licence a Rational Choice?	The Journal of Industrial Economics	2007	55	4	19
Journal Article	Garzarelli,Giampaolo; Limam,Yasmina Reem; Thomassen,Bj?rn	Open source software and economic growth: A classical division of labor perspective	Information Technology for Development	2008	14	2	12
Conference Proceedings	Raasch,C.; Herstatt,C.; Abdelkafi,N.	Creating Open Source Innovation: Outside the software industry	Management of Engineering & Technology, 2008. PICMET 2008. Portland International Conference on	2008			6
Journal Article	Aoki,Reiko; Schiff,Aaron	Promoting access to intellectual property: patent pools, copyright collectives, and clearinghouses	R&D Management	2008	38	2	29
Journal Article	Chen,Weibing; Li,Jingyue; Ma,Jianqiang;Conradi,Reidar; Ji,Junzhong; Liu,Chunnian	An empirical study on software development with open source components in the chinese software industry	Software Process: Improvement and Practice	2008	13	1	26
Journal Article	COLEMAN,GABRIELLA	CODE IS SPEECH: Legal Tinkering, Expertise, and Protest among Free and Open Source Software Developers	Cultural Anthropology	2009	24	3	52
Conference Proceedings	Alspaugh,T. A.; Asuncion,H. U.; Scacchi,W.	Analyzing software licenses in open architecture software systems	Emerging Trends in Free/Libre/Open Source Software Research and Development, 2009. FLOSS '09. ICSE Workshop on	2009			23
Journal Article	Coughlan, S., Katz,A.	Introducing The Risk Grid	International Free and Open Source Software Law Review	2009	1	1	
Journal Article	Henley,M.	Jacobsen v Katzer and Kamind Associates . an English legal perspective	International Free and Open Source Software Law Review	2009	1	1	
Journal Article	Kemp,R.	Towards Free/Libre Open Source Software (.FLOSS.) Governance in the Organisation	International Free and Open Source Software Law Review	2009	1	2	
Journal Article	Rosen,L.	Bad Facts Make Good Law: The Jacobsen Case and Open Source	International Free and Open Source Software Law Review	2009	1	1	
Journal Article	Sheppard,S.	Balancing Free with IP: If Open Source Solutions Become De Facto Standards Could Competition Law Start To Bite?	International Free and Open Source Software Law Review	2009	1	2	
Journal Article	Van den Brande,Y.	The Fiduciary Licence Agreement: Appointing legal guardians for Free Software Projects	International Free and Open Source Software Law Review	2009	1	1	

Reference Type	Authors, Primary	Title Primary	Periodical Full	Pub Year	Volume	Issue	Cited
Journal Article	Wong, C., Kreps,J.	Collaborative Approach: Peer-to-Patent and the Open Source Movement	International Free and Open Source Software Law Review	2009	1	1	
Journal Article	Colazo,Jorge; Fang,Yulin	Impact of license choice on Open Source Software development activity	Journal of the American Society for Information Science and Technology	2009	60	5	43
Conference Proceedings	Alspaugh,T. A.; Asuncion,H. U.; Scacchi,W.	Intellectual Property Rights Requirements for Heterogeneously-Licensed Systems	Requirements Engineering Conference, 2009. RE '09. 17th IEEE International	2009			29
Conference Proceedings	Di Penta,M.; German,D. M.	Who are Source Code Contributors and How do they Change?	Reverse Engineering, 2009. WCRE '09. 16th Working Conference on	2009			11
Conference Proceedings	German,D. M.; Hassan,A. E.	License integration patterns: Addressing license mismatches in component-based development	Software Engineering, 2009. ICSE 2009. IEEE 31st International Conference on	2009			55
Conference Proceedings	HongBo Xu; HuiHui Yang; Dan Wan; Jiangping Wan	The Design and Implement of Open Source License Tracking System	Computational Intelligence and Software Engineering (CiSE), 2010 International Conference on	2010			0
Conference Proceedings	Perry,M.; Margoni,T.	FLOSS for the Canadian Public Sector: Open Democracy	Digital Society, 2010. ICDS '10. Fourth International Conference on	2010			1
Journal Article	Lundell,Bj?rn; Lings,Brian; Lindqvist,Edvin	Open source in Swedish companies: where are we?	Information Systems Journal	2010	20	6	14
Journal Article	Anderson, H., Dare,T.	Passport Without A Visa: Open Source Software Licensing and Trademarks	International Free and Open Source Software Law Review	2010	1	2	
Journal Article	Mitchell QC,I.	Back To The Future: Hinton v Donaldson, Wood and Meurose (Court of Session, Scotland, 28th July, 1773)	International Free and Open Source Software Law Review	2010	1	2	
Journal Article	von Willebrand,M.	A look at EDU 4 v. AFPA, also known as the .Paris GPL-case.	International Free and Open Source Software Law Review	2010	1	2	
Journal Article	Webbink,M.	Packaging Open Source	International Free and Open Source Software Law Review	2010	1	2	
Journal Article	Brown,N.	GNU GPL 2.0 and 3.0: obligations to include license text, and provide source code	International Free and Open Source Software Law Review	2010	2	1	
Journal Article	Johnny, O., Miller, M., Webbink,M.	Copyright in Open Source Software - Understanding the Boundaries	International Free and Open Source Software Law Review	2010	2	1	
Journal Article	Piana,C.	Italian Constitutional Court gives way to Free-Software friendly laws	International Free and Open Source Software Law Review	2010	2	1	
Journal Article	Sinclair,A.	Licence Profile: BSD	International Free and Open Source Software Law Review	2010	2	1	
Journal Article	von Willebrand, M., Partanen,M.	Package Review as a Part of Free and Open Source Software Compliance	International Free and Open Source Software Law Review	2010	2	1	
Conference Proceedings	Di Penta,M.; German,D. M.; Antoniol,G.	Identifying licensing of jar archives using a code-search approach	Mining Software Repositories (MSR), 2010 7th IEEE Working Conference on	2010			11
Conference Proceedings	German,D. M.; Di Penta,M.; Davies,J.	Understanding and Auditing the Licensing of Open Source Software Distributions	Program Comprehension (ICPC), 2010 IEEE 18th International Conference on	2010			15
Conference Proceedings	Lindman,J.; Paajanen,A.; Rossi,M.	Choosing an Open Source Software License in Commercial Context: A Managerial Perspective	Software Engineering and Advanced Applications (SEAA), 2010 36th EUROMICRO Conference on	2010			4
Conference Proceedings	Di Penta,M.; German,D. M.; Gueheneuc,Y. -G; Antoniol,G.	An exploratory study of the evolution of software licensing	Software Engineering, 2010 ACM/IEEE 32nd International Conference on	2010	1		25

Reference Type	Authors, Primary	Title Primary	Periodical Full	Pub Year	Vol-ume	Issue	Cited
Conference Proceedings	Hongyu Zhang; Bei Shi; Lu Zhang	Automatic checking of license compliance	Software Maintenance (ICSM), 2010 IEEE International Conference on	2010			1
Journal Article	Scotchmer,Suzanne	Openness, Open Source, and the Veil of Ignorance	The American Economic Review	2010	100	2	6
Conference Proceedings	Yamakami,T.	A two-dimensional classification model of OSS: Towards successful management of the evolution of OSS	Advanced Communication Technology (ICACT), 2011 13th International Conference on	2011			0
Journal Article	Fitzgerald,B.	Open Source Software: Lessons from and for Software Engineering	Computer	2011	44	10	2
Conference Proceedings	Khanjani,A.; Sulaiman,Riza	The process of quality assurance under open source software development	Computers & Informatics (ISCI), 2011 IEEE Symposium on	2011			1
Journal Article	Bain,M.	Software Interactions and the GPL	International Free and Open Source Software Law Review	2011	2	2	
Journal Article	Brock,A.	Project Harmony: Inbound transfer of rights in FOSS Projects	International Free and Open Source Software Law Review	2011	2	2	
Journal Article	Dolmans, M., Piana,C.	A Tale of Two Tragedies . A plea for open standards	International Free and Open Source Software Law Review	2011	2	2	
Journal Article	Paapst,M.	Affirmative action in procurement for open standards and FLOSS	International Free and Open Source Software Law Review	2011	2	2	
Journal Article	Shemtov,N.	Software Patents and Open Source Models in Europe: Does the FOSS community need to worry about current attitudes at the EPO?	International Free and Open Source Software Law Review	2011	2	2	
Journal Article	Sinclair,A.	Licence Profile: Apache License, Version 2.0	International Free and Open Source Software Law Review	2011	2	2	
Journal Article	Aliprandi,S.	Interoperability and open standards: the key to a real openness	International Free and Open Source Software Law Review	2011	3	1	
Journal Article	Engelfriet,A.	Open source licensing notices in Web applications	International Free and Open Source Software Law Review	2011	3	1	
Journal Article	Freeman,A.	Patentable Subject Matter: The View from Europe	International Free and Open Source Software Law Review	2011	3	1	
Journal Article	Mitchell QC, I., Mason,S.	Compatibility Of The Licensing Of Embedded Patents With Open Source Licensing Terms	International Free and Open Source Software Law Review	2011	3	1	
Journal Article	Lichtenthaler,Ulrich	The evolution of technology licensing management: identifying five strategic approaches	R&D Management	2011	41	2	15
Conference Proceedings	Nakagawa,E. Y.; Maldonado,J. C.	Towards the Open Source Reference Architectures	Software Components, Architectures and Reuse (SBCARS), 2011 Fifth Brazilian Symposium on	2011			0
Journal Article	Lindman,J.; Rossi,M.; Puustell,A.	Matching Open Source Software Licenses with Corresponding Business Models	Software, IEEE	2011	28	4	10
Journal Article	Monden,A.; Okahara,S.; Manabe,Y.; Matsumoto,K. -i	Guilty or Not Guilty: Using Clone Metrics to Determine Open Source Licensing Violations	Software, IEEE	2011	28	2	7
Conference Proceedings	Kuang-Yuan Huang; Namjoo Choi	Relating and Clustering Free/Libre Open Source Software Projects and Developers: A Social Network Perspective	System Sciences (HICSS), 2011 44th Hawaii International Conference on	2011			2
Journal Article	Corbett,Susan	Creative Commons Licences, the Copyright Regime and the Online Community: Is there a Fatal Disconnect?	The Modern Law Review	2011	74	4	2
Conference Proceedings	Ziemer,S.	On the adoption of open source software in aeronautics	Aerospace Conference, 2012 IEEE	2012			0
Journal Article	Gaff,B. M.; Ploussios,G. J.	Open Source Software	Computer	2012	45	6	0
Journal Article	Aliprandi,S.	Open licensing and databases	International Free and Open Source Software Law Review	2012	4	1	

Reference Type	Authors, Primary	Title Primary	Periodical Full	Pub Year	Volume	Issue	Cited
Journal Article	Greenbaum,E.	The GPL .Liberty or Death!. Clause: An Israeli Case Study	International Free and Open Source Software Law Review	2012	4	1	
Journal Article	Katz,A.	Towards a Functional Licence for Open Hardware	International Free and Open Source Software Law Review	2012	4	1	
Journal Article	Meeker,H.	The Gift that Keeps on Giving . Distribution and Copyleft in Open Source Software License	International Free and Open Source Software Law Review	2012	4	1	
Journal Article	Tiller,R.	Initial thoughts on Mayo v. Prometheus and software patents	International Free and Open Source Software Law Review	2012	4	1	
Conference Proceedings	Alspaugh,T. A.; Scacchi,W.; Kawai,R.	Software licenses, coverage, and subsumption	Requirements Engineering and Law (RELAW), 2012 Fifth International Workshop on	2012			0
Conference Proceedings	Tatsubori,M.; Gangadharan,G. R.	Service Commons – Serve and Serve Alike: Applying the Creative Commons Spirit to Web Services	Services Computing (SCC), 2012 IEEE Ninth International Conference on	2012			0
Conference Proceedings	Forbes,C.; Keivanloo,I.; Rilling,J.	When open source turns cold on innovation . The challenges of navigating licensing complexities in new research domains	Software Engineering (ICSE), 2012 34th International Conference on	2012			1
Conference Proceedings	Mathur,Arunesh; Choudhary,Harshal; Vashist,Priyank; Thies,William; Thilagam,Santhi	An Empirical Study of License Violations in Open Source Projects	Software Engineering Workshop (SEW), 2012 35th Annual IEEE	2012			0
Journal Article	German,D. M.; Di Penta,M.	A Method for Open Source License Compliance of Java Applications	Software, IEEE	2012	29	3	2
Journal Article	van Holst,W.	Copyleft, -right and the case law on APIs on both sides of the Atlantic	International Free and Open Source Software Law Review	2013	5	1	
Conference Proceedings	Lokhman,Alexander; Mikkonen,Tommi; Hammouda,Imed; Kazman,Rick; Chen,Hong-Mei	A Core-Periphery-Legality Architectural Style for Open Source System Development	System Sciences (HICSS), 2013 46th Hawaii International Conference on	2013			0
Journal Article	Vasudeva,Vikrant Narayan	A Relook at Sui Generis Software Protection Through the Prism of Multi?Licensing	The Journal of World Intellectual Property	2013	16	1-2	0